لغات برمجة C++

منشورات جامعة حلب
كلية الهندسة الكهربائية والإلكترونية

لغات برمجة C++

المهندسة

أمل منا

مدرسة متفرغة في وزارة التعليم العالي

لطلاب السنة الثانية
اتصالات، قيادة كهربائية، نظم القدرة

مديرية الكتب والمطبوعات الجامعية
١٤٢٧هـ - ٢٠٠٦م

الفهرس

الموضـــــــــــــــــــوع رقم الصفحة

المقدمة ١٧

الفصل الأول : مدخل إلى الحاسوب ولغات البرمجة

1-1- نبذة تاريخية ٢١
1-2- الحاسوب ومعالجة المعطيات ٢١
1-3- مبدأ عمل الحاسوب وأقسامه ٢٣
1-4- البرنامج ولغات البرمجة ٢٦
1-4-1- لغات منخفضة المستوى ٢٦
1-4-1-1- لغة الآلة ٢٦
1-4-1-2- اللغة التجميعية ٢٧
1-4-2- اللغات عالية المستوى ٢٧
1-5- تمثيل المعلومات في الحاسوب ٢٩
1-5-1- نظام الترميز الثنائي أو الترميز الرقمي ٢٩
1-5-2- نظام الترميز الثماني ٣٠
1-5-3- نظام الترميز الست عشري ٣٠
1-5-4- نظام الترميز الثنائي العشري B.C.D. ٣١
1-6- التحويل من نظام الترميز العشري إلى نظام الترميز الثنائي ٣١
1-7- التحويل من نظام الترميز الثنائي إلى نظام الترميز العشري ٣٣
1-8- المراحل المطلوبة لوضع برنامج للحاسوب ٣٣

الموضوع رقم الصفحة

الفصل الثاني : الخوارزميات ومخططات سير العمليات

2-1- مقدمة ٣٧
2-2- أنواع الخوارزميات ٣٨
  2-2-1- الخوارزمية غير الحسابية ٣٨
  2-2-2- الخوارزمية الحسابية ٣٨
2-3- طرق كتابة الخوارزمية ٣٩
  2-3-1- استخدام اللغة الطبيعية في صياغة الخوارزمية ٣٩
  2-3-2- استخدام الطريقة الرمزية في صياغة الخوارزمية ٤٠
  2-3-3- استخدام الطريقة البيانية في صياغة الخوارزمية ٤٠
2-4- أنواع مخططات سير العمليات ٤٢
  2-4-1- مخططات تتابعية (تسلسلية) ٤٢
  2-4-2- مخططات تفرعية ٤٣
  2-4-3- مخططات تكرارية ٤٥
2-5- تمارين ٤٩

الفصل الثالث : مفاهيم أساسية في لغة البرمجة ++C

3-1- نبذة تاريخية عن لغة ++C ٥٥
3-2- أساسيات اللغة البرمجية ٥٨
  3-2-1- الرموز الأساسية (C++ symbols) ٥٨
  3-2-2- الكلمات المحجوزة ٥٩

الموضـــــــــــوع رقم الصفحة
3-2-3- الأسماء التعريفية ٦٠
3-2-4- المتغيرات ٦٠
3-2-5- الثوابت ٦٢
3-2-6- المعطيات غير العددية (الرمزية) ٦٣
3-3- التعبير ٦٤
3-4- تعليمة التعيين ٦٧
3-4-1- تعليمة التعيين المتعدد ٦٧
3-5- عوامل التعيين المركبة ٦٩
3-6- عوامل التزايد والتناقص ٦٩
3-7- استخدام عوامل التزايد والتناقص ضمن التعبير ٧٠
3-8- أولوية تنفيذ عوامل لغة C++ في تعابير حسابية ومنطقية ٧٣
3-9- برنامج بسيط في لغة C++ ٧٦
3-10- تمارين ٨٠

الفصل الرابع : تعليمات الإدخال والإخراج

4-1- تعليمة التصريح عن البيانات ٨٥
4-2- إدخال البيانات باستخدام cin>> ٨٧
4-3- إخراج البيانات في برنامج C++ وتنسيقها ٨٨
4-3-1- تعليمة الإخراج cout<< ٨٨
4-3-2- تنسيق الخرج باستخدام رموز تتابع الهروب ٩٠

الموضوع رقم الصفحة
4-3-3- استخدام التابع setw(N) ٩٢
4-3-4- التحكم بشكل إخراج الأعداد الحقيقية ٩٣
4-4- تمارين ٩٥

الفصل الخامس : تعليمات التحكم واتخاذ القرار

5-1- تعليمة الشرط if ١٠١
5-2- تعليمة الشرط if..else ١٠٦
5-3- تعليمة التعبير الشرطي ١٠٨
5-4- الشروط المركبة ١٠٩
5-5- الشروط المتداخلة ١١١
5-6- تعليمة الاختيار المتعدد switch ١٢١
5-7- تمارين ١٢٦

الفصل السادس : التكرار وحلقات التكرار

6-1- استخدام تعليمة الانتقال غير المشروط goto ١٤١
6-2- البنية التكرارية while ١٤٣
6-3- البنية التكرارية do..while ١٤٦
6-4- البنية التكرارية for ١٤٩
6-5- التكرار اللانهائي ١٥٢
6-5-1- حلقة for اللانهائية ١٥٢
6-5-2- حلقة while اللانهائية ١٥٣

الموضـــــــــــــــــــــــــــوع رقم الصفحة

6-6- تعليمة break ١٥٤

6-7- تعليمة الاستمرار Continue ١٥٦

6-8- الحلقات المتداخلة nested loop ١٥٧

6-8-1- حلقات for المتداخلة ١٥٨

6-8-2- حلقات while المتداخلة ١٥٩

6-9- تمارين ١٦١

الفصل السابع : المصفوفات والسلاسل

7-1- مقدمة ١٧٧

7-2- التصريح عن المصفوفة ١٧٨

7-3- المصفوفة أحادية البعد ١٨٠

7-3-1- إعطاء قيم ابتدائية لعناصر مصفوفة أحادية ١٨١

7-3-2- إدخال وإخراج قيم عناصر المصفوفة أحادية البعد ١٨٢

7-3-3- فرز وترتيب المصفوفات ١٨٩

7-3-4- البحث في المصفوفات ١٩٢

7-4- المصفوفة ثنائية البعد ١٩٤

7-4-1- إعطاء قيم ابتدائية لعناصر مصفوفة ثنائية البعد ١٩٥

7-4-2- إدخال وإخراج قيم عناصر المصفوفة ثنائية البعد ١٩٦

7-5- السلاسل ٢٠٠

7-5-1- التصريح عن السلاسل الحرفية ٢٠٠

الموضوع رقم الصفحة

7-5-2- إعطاء قيم ابتدائية للسلاسل الحرفية ٢٠٠

7-5-3- إدخال وإخراج قيم عناصر السلسلة الحرفية ٢٠١

7-6- مصفوفات السلاسل ٢٠٦

7-7- مكتبة توابع التعامل مع سلاسل المحارف ٢٠٧

7-7-1- التابع strlen (s) ٢٠٧

7-7-2- التابع strcmp و strncmp ٢٠٨

7-7-3- التابع strcat و strncat ٢١٠

7-7-4- التابع strcpy و strncpy ٢١٢

7-7-5- التابع strtok ٢١٥

7-8- تمارين ٢١٨

الفصل الثامن : التوابع

8-1- مقدمة ٢٣١

8-2- أهمية استخدام التوابع ٢٣٢

8-3- أنواع التوابع ٢٣٣

8-4- مكتبة التوابع القياسية ٢٣٤

8-5- توابع من تصميم المبرمج ٢٣٧

8-5-1- الإعلان عن التابع ٢٣٧

8-5-2- استدعاء التابع ٢٤٠

8-5-3- تعريف التابع ٢٤١

رقم الصفحة	الموضـــــــــــــــــــــــــــــــــــــوع
٢٤٥	8-6- التعاودية recursion
٢٤٩	8-7- المعرفات ونوع التخزين ومجال الرؤية
٢٥٥	8-8- العملية الأحادية لتحديد مجال الرؤية Unary scope resolution operator
٢٥٧	8-9- التحميل الزائد للتوابع
٢٦٠	8-10- التابع inline
٢٦٢	8-11- البارامترات الافتراضية Default parameters
٢٦٦	8-12- تمارين

الفصل التاسع : المراجع والمؤشرات

رقم الصفحة	الموضوع
٢٧١	9-1- المرجع
٢٧١	9-1-1- التصريح عن المرجع
٢٧٣	9-1-2- تمرير بارامترات التوابع بوساطة المراجع
٢٨٠	9-1-3- تمرير البارامترات بمرجع ثابت
٢٨٤	9-2- المؤشرات
٢٨٤	9-2-1- التصريح عن المؤشرات وإعطائها قيماً ابتدائية
٢٨٨	9-2-2- عامل التعيين غير المباشر
٢٨٨	9-2-3- تمرير بارامترات التوابع بوساطة المؤشرات
٢٩٠	9-2-4- استخدام العامل new و delete
٢٩٤	9-2-5- التعابير والعمليات الحسابية على المؤشرات
٢٩٥	9-2-5-1- علاقة التعيين للمؤشرات

الموضوع رقم الصفحة

9-2-5-2- العمليات الحسابية على المؤشرات ٢٩٦
9-2-5-3- مقارنة المؤشرات ٢٩٨
9-2-6- العلاقة بين المصفوفات والمؤشرات ٢٩٩
9-2-6-1- المؤشرات والمصفوفات الأحادية البعد ٢٩٩
9-2-6-2- المؤشرات والمصفوفات ثنائية البعد ٣٠٥
9-2-7- السلاسل والمؤشرات ٣٠٧
9-2-7-1- السلاسل كبارامترات للتوابع ٣٠٨
9-2-8- استخدام الواصف const مع المؤشرات ٣١١
9-2-8-1- المؤشر غير الثابت لمعطيات غير ثابتة ٣١١
9-2-8-2- المؤشر غير الثابت لمعطيات ثابتة ٣١٢
9-2-8-3- المؤشر الثابت لمعطيات غير ثابتة ٣١٤
9-2-8-4- المؤشر الثابت لمعطيات ثابتة ٣١٦
9-2-9- مصفوفات المؤشرات ٣٢١
9-2-10- مؤشرات لمؤشرات ٣٣٢
9-2-11- التوابع التي ترجع مؤشرات ٣٣٣
9-2-12- المؤشرات إلى توابع ٣٣٦
9-2-13- مشاكل المؤشرات ٣٤١
9-3- تمارين ٣٤٣

الموضوع رقم الصفحة

الفصل العاشر : أنواع البيانات المعرفة من قبل المستخدم

10-1- تعليمة typedef ٣٨٣

10-2- تعليمة نوع البيانات التعدادي enumeration ٣٨٤

  10-2-1- إنشاء متغيرات تعدادية واستعمالها ٣٨٦

10-3- البنية structure ٣٨٨

  10-3-1- تعريف البنية ٣٨٨

  10-3-2- تعريف متغير من نوع البنية ٣٩٠

  10-3-3- الوصول إلى أعضاء البنية ٣٩١

  10-3-4- إعطاء قيم ابتدائية لمتغير البنية ٣٩٣

  10-3-5- تعيين قيمة متغير بنية إلى متغير آخر ٣٩٣

  10-3-6- مصفوفات البنية ٣٩٣

  10-3-7- تمرير البنية إلى التوابع بالقيمة ٣٩٧

  10-3-8- الحصول على بنية من تابع ٣٩٨

  10-3-9- البنى المتداخلة ٤٠١

  10-3-10- التصريح عن مؤشرات لبنية ٤٠٥

  10-3-11- تمرير البنية إلى التوابع بالعناوين ٤٠٦

10-4- تمارين ٤٠٨

الفصل الحادي عشر : الأصناف

11-1- برمجة الكيان ٤١٣

المـــوضـــــــــــــــــــــوع رقم الصفحة

11-1-1- مقدمة ٤١٣
11-1-2- الأصناف ٤١٤
11-1-3- التوارث ٤١٦
11-1-4- التوابع والأدوات المتعددة الأغراض ٤١٧
11-2- الأصناف ٤١٨
11-2-1- تعريف الصنف ٤١٩
11-2-2- مجال رؤية الصنف ٤٢٣
11-2-3- الوصول إلى أعضاء الصنف ٤٢٣
11-2-4- توجيه ما قبل الترجمة ٤٢٩
11-2-5- تعليمات التعيين مع الكائنات ٤٣٢
11-2-6- توابع البناء وتوابع الهدم في الصنف ٤٣٥
11-2-6-1- التحميل الزائد للتابع الباني والمتحولات الافتراضية ٤٤٢
11-2-6-2- استخدام القيم الافتراضية للباراميترات في التابع الباني ٤٤٥
11-2-6-3- التابع الباني الناسخ ٤٤٦
11-2-7- توابع الوصول ٤٥٤
11-2-8- التوابع الأعضاء الخاصة ٤٥٥
11-2-9- التوابع الأعضاء العامة في الصنف ٤٥٦
11-2-9-1- التوابع الأعضاء العامة في الصنف بدون باراميترات ٤٥٧
11-2-9-2- التوابع الأعضاء العامة في الصنف مع باراميترات ٤٥٨

المـــوضــــــــــــــــــــوع رقم الصفحة

11-2-9-3- التوابع الأعضاء العامة التي تغير في قيم البيانات الأعضاء ٤٦٠
11-2-9-4- التوابع الأعضاء العامة التي تجري عمليات حسابية على البيانات الأعضاء ٤٦١
11-2-9-5- إعادة قيمة من التوابع الأعضاء ٤٦٣
11-2-9-6- استدعاء التوابع الأعضاء بالمرجع ٤٦٧
11-2-10- البيانات الأعضاء الخاصة في الصنف ٤٧٠
11-2-10-1- استخدام المصفوفة الأحادية كعضو بيانات في الصنف ٤٧٠
11-2-10-2- استخدام السلسلة كعضو بيانات في الصنف ٤٧٥
11-2-10-3- استخدام مصفوفة سلسلية كعضو بيانات في الصنف ٤٧٧
11-2-10-4- استخدام المؤشر كعضو بيانات في الصنف ٤٧٨
11-2-11- التوابع الصديقة ٤٨١
11-2-12- قالب التابع ٤٨٤
11-2-13- التوارث ٤٨٧
11-3- تمارين ٥٠٠
مسائل برمجية محلولة ٥١٠
الملاحق ٥٤٣
الملحق (A) شفرات ASCII ٥٤٥
الملحق (B) الكلمات المفتاحية في لغة ++C ٥٥١
الملحق (C) العمليات في لغة ++C ٥٥٥

الموضـــــــــــــــــــــــوع	رقم الصفحة
الملحق (D) الأنواع في لغة ++C	٥٥٧
الملحق (E) توابع لغة ++C مسبقة التعريف	٥٥٨
المراجع العربية	٥٦٥
المراجع الأجنبية	٥٦٧
المصطلحات العلمية	٥٦٩

المقدمة

لقد بدأت لغة البرمجة ++C تحل محل لغة البرمجة C إذ إنها أضحت الخيار الأول للمبرمجين في الصناعة في بداية الثمانينات. وقد قدمت هذه اللغة العديد من الإمكانات التي تسمو بلغة C وتعلو بها، والأهم هو بقدرات البرمجة غرضية التوجه (Object oriented programming) . إذ إن الأغراض (object) من المكونات البرمجية القابلة للاستخدام أكثر من مرة، وهي قادرة على تمثيل العناصر من العالم الحقيقي المحيط بنا، وهذا الغرض (الكائن) يدمج البيانات والتوابع التي تعمل على تلك البيانات في كينونة واحدة مكتفية ذاتياً (أي يشكل وحدة برمجية متكاملة).

إن قوة البرمجة غرضية التوجه هي في صفاتها الثلاث :

التصنيف (الإخفاء) ((classes (Encapsulation)) والتوارث (Inheritance) والتعددية الشكلية (Polymorphism). وهذا ما جعل لغة ++C لغة مثالية وذات قوة عالية حيث تعطي المبرمج حرية الاختيار وتقلل من حجم البرنامج المستخدم، حيث يستطيع الاستفادة من كثير من التوابع والأصناف (classes) المكتوبة.

يهدف هذا الكتاب إلى تعريف الطالب (ذي المعلومات القليلة أو من دون خبرة برمجية مسبقة) بأسس ومبادئ حل المسائل حاسوبياً، والتي يمكن تقسيمها إلى مرحلتين:

المرحلة الأولى : وهي فهم المسألة وتحليلها وتطوير استراتيجية أو خوارزمية تعطي الجواب. حيث تقسم الخوارزمية إلى سلسلة من الخطوات أو العمليات البسيطة مع ضرورة الانتباه إلى الحالات الخاصة .

المرحلة الثانية : تكون بكتابة برنامج ينفذ العمليات المطلوبة ومن ثم يعطي الحل.

إن الاهتمام الأساسي لهذا الكتاب هو في المرحلة الثانية حيث يوضح كيفية كتابة برامج بلغة البرمجة C++ التي تتمتع بقدرتها على عرض المفاهيم البرمجية التقليدية بالإضافة إلى التوجه الجديد للبرمجة والمسمى بالبرمجة غرضية التوجه.

لايمكن أن يكون هذا الكتاب شاملاً ويبين خبايا اللغة المتقدمة، ولكنه يتعرض بشكل مبسط وموضح بأمثلة عديدة تم جمعها واختبارها على مدى السنوات الثماني التي درس فيها هذا المقرر وهي منتقاة من مراجع مختلفة وفي اختصاصات عديدة.

يتضمن هذا الكتاب فهرساً ومقدمة وملاحق وبعض المصطلحات العلمية والمراجع واحد عشر فصلاً تشرح لغة C++ حيث :

يقدم **الفصل الأول** مقدمة عن الحاسوب، ومبدأ عمله وأقسامه، وتعريفاً بمفهوم البرنامج ولغات البرمجة، وتمثيل المعلومات في الحاسوب والمراحل المطلوبة لوضع برنامج للحاسوب.

يتعرض **الفصل الثاني** لمفهوم الخوارزمية ومخصصات سير العمل فيحدد أنواع الخوارزميات وطرق كتابتها ورموز مخططات سير العمل وأنواعها.

الفصل الثالث : يشرح المفاهيم الأساسية في لغة البرمجة C++ (الرموز، الكلمات المفتاحية، الأسماء التعريفية، المتغيرات ، الثوابت، المعطيات العددية)، التعابير، تعليمة التعيين، عوامل التعيين المركبة، عوامل التزايد والتناقص، أولوية تنفيذ عوامل لغة C++ في التعابير الحسابية والمنطقية، ثم يتعرض لبرنامج بسيط بلغة C++.

الفصل الرابع : يوضح طرق إدخال البيانات إلى الحاسوب والتحكم بها وطرق إخراج المعلومات والتحكم بشكل المخرجات.

الفصل الخامس : يتناول تعليمات التحكم واتخاذ القرار.

الفصل السادس : يتعرض لتعليمات التكرار وحلقات التكرار.

الفصل السابع : يتضمن شرحاً للمصفوفات العددية الأحادية وثنائية البعد وكذلك السلاسل الحرفية .

الفصل الثامن : يتناول التوابع وآلية ربطها مع بعضها ببعض ومع البرنامج الرئيسي.

الفصل التاسع : يتعرض لمفهوم المراجع والمؤشرات وطرق استخدامها والتصريح عنها وتطبيقاتها مع السلاسل الحرفية والمصفوفات المكونة من هذه السلاسل. وكذلك كيفية استخدامها مع المصفوفات العددية وكيفية استدعاء التوابع بوساطة العنوان وبوساطة المرجع والمتغيرات المرجعية، كما يبحث في كيفية تمرير المصفوفات إلى التوابع.

الفصل العاشر : يتضمن شرحاً عن أنواع البيانات المعرفة من قبل المستخدم مثل enum , typedef والبنى (struct) وتطبيقاتها .

الفصل الحادي عشر : يتناول الأصناف وكيفية التصريح عنها وإنشاء الكيانات من المتغيرات والتوابع الأعضاء، ومعالجتها واستخدامها لبناء النظم البرمجية المختلفة.

يتوجه هذا الكتاب إلى طلاب السنة الثانية قسم هندسة الاتصالات وهندسة القيادة الكهربائية وهندسة نظم القدرة ويمكن أن يفيد منه طلاب الأقسام الأخرى في كلية الهندسة الكهربائية والإلكترونية وكذلك طلاب كلية الهندسة المعلوماتية.

آمل أن يحقق هذا الكتاب الهدف المرجو منه في تنمية قدرة الطالب على فهم أسس البرمجة والإمكانات المتوافرة في اللغة في حل مجموعة كبيرة من التطبيقات العلمية والهندسية.

حلب في ــ ٢٠٠٦م

المهندسة أمل منا

الفصل الأول

مدخل إلى الحاسوب ولغات البرمجة

Introduction to Computer & Programming Languages

1-1- نبذة تاريخية:

قدم الفكر الإنساني في الأربعينات والخمسينات من القرن العشرين ابتكاراً يُعد بحق من أعظم ما قدمته الحضارة الإنسانية، ويماثل في أهميته وفاعليته وتأثيره في المجتمع أهمية اختراع الطباعة وهو اختراع الحاسوب (Computer) . ولقد كان هذا الاختراع ثمرة جهود كبيرة لعلماء بارزين أرادوا الوصول إلى مكننة العمل الفكري بعد أن تمت مكننة العمل الجسدي نتيجة الثورة الصناعية في القرن الثامن عشر.

إن أول آلة ميكانيكية تقوم بالحسابات آلياً تم بناؤها من قبل العالم الفرنسي الشهير بليزباسكال Blaise Pascal الذي أنجز عام 1642 آلته القادرة على إجراء عمليتي الجمع والطرح آلياً. وطوّر العالم ليبنز W.L.Leibniz في عام 1671 آلة باسكال لتقوم بالضرب والقسمة أيضاً.

ثم كان الحاسوب المسمى Eniac وهو أول الحواسيب الإلكترونية المتعددة الأغراض والذي اعتمد في بنائه على أفكار العالم الرياضي فون نيومان Von Neumann الهنغاري الأصل في جامعة بنسلفانيا بين عامي 1942 و 1946، وبذلك بدأت حقبة جديدة متسارعة تطورت فيها تقنيات تصنيع الحواسيب، وهي مازالت مستمرة إلى يومنا هذا، مما أدى إلى انتشار استخدام الحواسيب في شتى النشاطات الاجتماعية والإنسانية وفجر عالماً جديداً هو عالم المعلومات.

1-2- الحاسوب ومعالجة المعطيات:

اشتق الاسم "حاسوب" على وزن التكثير من المصدر "حساب" بمعنى الآلة كثيرة الحساب ، ولكن العمل الحقيقي اليوم لهذا الجهاز بعيد كل البعد في معظم

تطبيقاته عن الحساب الذي لايتعدى كونه أحد أنواع الأعمال التي يقوم بها وليس بالضرورة أهمها. ويمكن اعتبار الحاسوب بالدرجة الأولى جهازاً لتخزين المعطيات (البيانات Data) ومعالجتها ونقلها وتداولها بمفهوم المعلومات العريض الشامل وبأصنافها المختلفة الكثيرة.

ومن استعراض أنواع المعطيات التي يعالجها الحاسوب نستطيع القول إنها يمكن أن تأخذ أحد الأشكال التالية:

1- رقماً من الأرقام كما هي الحال في المعطيات ذات الطابع المالي أو العلمي أو الهندسي أو الإحصائي.

2- جملة مكتوبة مكونة من حروف أو نص كامل كما هو الحال عند التعامل مع المعلومات الشخصية للأفراد أو في حالة نشر النصوص وحفظها والتعامل معها وتعديلها.

3- شكلاً بيانياً أو صورة أو مخططاً يجري التعامل معه ومعالجته وإظهاره.

4- عبارة صوتية منطوقة يتم التعرف عليها وفهمها وربما تحويلها إلى نص مكتوب آلياً.

5- إشارة تشابهية متناسبة مع قياس يصدر عن جهاز، أو إشارة منطقية تتحكم في أجهزة، كما هي الحال في الآلات الصناعية الحديثة والطائرات ونظم توزيع الطاقة الكهربائية والمياه.

وباعتبار أن الحاسوب أداة متعددة الأغراض، قادرة على إنجاز أعمال مختلفة ومتنوعة وبسرعة فائقة. فلابدّ عند محاولة القيام بوظيفة معينة، أو عمل محدد، باستخدام الحاسوب من وجود وسيلة تنقل إليه تفاصيل العمل المطلوب إنجازه، ويجري ذلك عادة بوساطة البرمجيات (Software) مما يتطلب من مستخدم الحاسوب أن يتعرف على بنية هذه البرمجيات وتفاصيلها بالإضافة إلى إلمام مبسط بآلية عمل الحاسوب بخطوطها العريضة.

إن العملية التي يقوم فيها الحاسوب بربط الحقائق المنفصلة مع بعضها تسمى بعملية المعالجة (Processing) وتسمى الوحدة المنفذة لهذه العملية في الحاسوب بالمعالج (Processor) أما النتائج التي نحصل عليها فتسمى بالمعلومات (Information).

1-3- مبدأ عمل الحاسوب وأقسامه:

يمكن مقارنة طريقة قيام الحاسوب بعملية معينة بتلك التي يقوم بها شخص ما بالعمل نفسه. ففي البدء يجب أن نحدد لهذا الشخص تفاصيل العمل المطلوب، أو بعبارة أخرى تزويده بالتعليمات التي تبين له كيفية الوصول إلى النتيجة المطلوبة ثم يجب أن نقدم له البيانات التي سيقوم باستخدامها، والحاسوب حتى يقوم بعمله يحتاج إلى ما يكافئ ذلك تماماً، أي هو يحتاج إلى مجموعة من التعليمات تحدد ما يجب أن يقوم به من عمليات، ومجموعة البيانات التي يستخدمها للحصول على النتائج. إذن لابدّ من إدخال البيانات إلى الحاسوب (Input) ومن ثم يقوم بمعالجتها (Processing) حتى نحصل على الخرج (Output) المطلوب سواء عملية أو نتيجة.

نسمي جميع الأجهزة التي تجعل الحاسوب قادراً على إنجاز العمليات الثلاث: إدخال – معالجة – إخراج بالتجهيزات الحاسوبية (Hardware) ، وتقسم هذه التجهيزات إلى الأقسام التالية:

1- أجهزة الإدخال : وتستخدم كوسيلة لإدخال المعطيات والتعليمات والأوامر من الوسط الخارجي إلى الحاسوب.

2- أجهزة الإخراج : وتستخدم كوسيلة لإخراج البيانات والمعطيات والنتائج من الحاسوب إلى الوسط الخارجي.

ويستخدم الحاسوب العديد من أجهزة الدخل والخرج وتختلف باختلاف البيانات المدخلة أو المخرجة، ومنها : الشاشة، ولوحة المفاتيح كأجهزة دخل وخرج للبيانات الحرفية (Character) والعددية (Number) ، والطابعات (Printer) كأجهزة خرج

لنوع البيانات والمعلومات نفسه ، منضدة الترقيم (Digitizer) وهي جهاز دخل بياني لقراءة إحداثيات النقاط آلياً إذا كانت المعلومات على شكل بياني، الراسم البياني (Plotter) لإظهار الرسوم والمنحنيات البيانية فهو بذلك جهاز خرج بياني، وحدات قيادة الأقراص اللينة (Floppy Disc Drive) ، ووحدات قيادة الأقراص المدمجة (CD Drive) كوحدات دخل وخرج.

3- وحدة المعالجة المركزية (Central Processing Unit – C.P.U) : وتقوم بإنجاز كل عمليات المعالجة والحسابات المختلفة.

4- وحدة التحكم : تقوم وحدة التحكم بتنظيم وتنسيق عمل جميع وحدات الحاسوب الأخرى وفقاً لخطة عمل جاهزة ومعدة سلفاً ومخزنة في ذاكرة الحاسوب على شكل تعليمات تؤخذ بالتسلسل واحدة بعد الأخرى ويجري تحليلها وتنفيذها بوساطة وحدة التحكم التي ترسل الإشارات الإلكترونية المناسبة إلى وحدة المعالجة الحسابية والمنطقية .(Arithmatic and Logical Unit – A.L.U) والتي هي عبارة عن وحدة فرعية من وحدة المعالجة المركزية (C.P.U) لتنفيذ العمليات المطلوبة، وتسمى وحدة الحساب والمنطق ووحدة التحكم بالمعالج .

5- أجهزة التخزين (Memory): وتستخدم لتخزين جميع البيانات والتعليمات بشكل دائم ومؤقت ولها عدة أنواع :

1) **الذاكرة المركزية** : وتتميز بالسرعة الكبيرة وتخزن فيها البرامج والمعلومات عند معالجتها ولها نوعان هما :

a- الذاكرة الثابتة للقراءة فقط (Read Only Memory_ROM) وهي ذاكرة دائمة تحتفظ بالمعلومات حتى بعد انقطاع التغذية الكهربائية عن الجهاز، ولكن لايمكن تعديلها، وتستخدم لتخزين البرامج والمعلومات الثابتة مثل برنامج إقلاع الحاسوب.

b- ذاكرة الوصول العشوائي (Random Access Memory - RAM) وهي ذاكرة مؤقتة لاتحتفظ بالمعلومات إلاّ أثناء تغذية الحاسوب بالطاقة الكهربائية، ويستطيع الحاسوب أن يكتب فيها وأن يقرأ منها المعلومات، ويقوم الحاسوب بنقل البرامج والمعلومات التابعة لها من الذاكرة الثانوية إلى هذه الذاكرة قبل الشروع بتنفيذها.

2) **التخزين الثانوي** (Secondary Storage) : ويتميز بكلفته الصغيرة بالمقارنة مع الذاكرة المركزية ويؤمن سعات تخزين أكبر بكثير مما يمكن أن توفره الذاكرة المركزية فقط، ولكن يحصل على هذه الميزة على حساب سرعة الوصول إلى المعلومات التي تكون أصغر بكثير من نظيرتها في الذاكرة المركزية. وتستخدم الأقراص الممغنطة والليزرية من أجل التخزين الثانوي.

تقاس عادة قدرات الحاسوب بالاستناد إلى ما يلي :

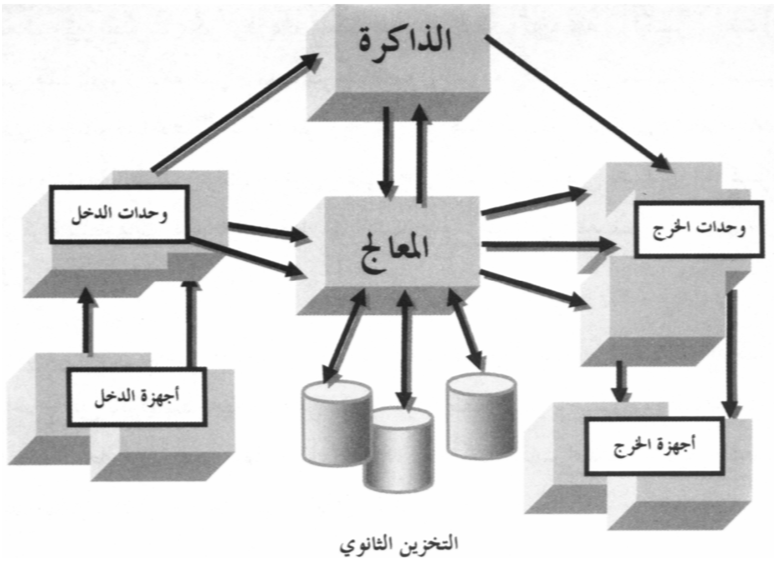
1- سرعة المعالجة (Processing speed) : وهو الزمن اللازم للحاسوب لإنجاز عملية واحدة كجمع عددين ، فمثلاً يستطيع الحاسوب إنجاز 10 ملايين عملية جمع في الزمن اللازم للإنسان لإغماض عينيه وفتحها.

2- زمن الوصول (Access time) : ويُعرف بأنه الزمن لتحديد مكان وجود المعلومة والحصول عليها.

3- معدل نقل المعلومات (Transfer rate) : ويُعرف بأنه عدد الرموز المنقولة ما بين موضعين خلال وحدة الزمن، مثلاً أكثر من 800000 رمز في الثانية ومعدل النقل في الذاكرة هو أعلى بكثير من معدله في وحدات الأقراص أو غيرها من الأجهزة.

4- قدرة التخزين : وتُعرف بأنها كمية المعلومات الممكن تخزينها في وحدة التخزين ولقد ازدادت في السنوات الأخيرة بشكل هائل وتقاس سعة التخزين بـ Kbyte أو Mbyte أو Gbyte حيث : الـ byte هو أصغر وحدة تخزين ويحوي 8.بت(bit).

ويمثل الشكل (1-1) التالي بنية الحاسوب العامة وتشتمل على أجهزة ووحدات الدخل والخرج.



الشكل (1-1) **بنية الحاسوب**

1-4- البرنامج ولغات البرمجة:

بالإضافة إلى التجهيزات التي تحدثنا عنها في الفقرة السابقة، ولإكمال نظام الحاسوب نحتاج إلى مجموعة من البرامج (Software) نستطيع بوساطتها إعطاء التعليمات إلى الحاسوب لإنجاز عمليات الإدخال والمعالجة والإخراج، ويُعرف البرنامج (Program) بأنه مجموعة من الأوامر مكتوبة بلغة الحاسوب أو بلغة يفهمها الحاسوب تؤدي إلى إنجاز عملية معينة أو حل مسألة محددة.

أما بالنسبة للغات الحاسوب فيمكن أن نميز ما يلي:

1-4-1- لغات منخفضة المستوى (Low Level Languages):

1-4-1-1- لغة الآلة (Machine Language):

تعتمد مفردات هذه اللغة على مبدأ الترقيم الثنائي (Binary Code) أي استخدام (0) أو (1) فقط ، وهي اللغة الوحيدة التي يفهمها الحاسوب مباشرة ويستطيع

تنفيذها من دون الحاجة إلى ترجمتها، ولكنها صعبة الكتابة إذ إن الإنسان لايملك الوقت والصبر الكافيين لحفظ وقراءة صفوف طويلة من الأرقام الثنائية، كما يصعب إصلاح واكتشاف أخطاء البرنامج المكتوب بلغة الآلة، وبما أن الأنواع المختلفة للحواسيب تتصف بمجموعة تعليمات مختلفة، فهذا يتطلب لغة آلة خاصة لكل نوعٍ من أنواعها، ولهذه الأسباب حاول الإنسان اللجوء لأنظمة عددية مختلفة لحزم الأرقام الثنائية كالنظام الثماني الذي يجمع كل ثلاثة أرقام ثنائية برقم واحد ثُماني، والنظام الست عشري الذي يجمع كل أربعة أرقام ثنائية برقم واحد ست عشري، وهذا يخفض عدد الأرقام اللازمة لتمثيل أمر برمجي، فمثلاً : إذا لزم لتنفيذ أمر ما 16 رقماً ثنائياً عندها يكفي كتابته بأربعة أرقام ستة عشرية.

لم تحل النظم العددية المشكلة كلياً إذ مازال البرنامج يكتب بدلالة أعداد لا كلمات مفهومة، ولحل هذه المشكلة جزئياً كانت اللغة التجميعية وهي لغة وسط ما بين لغة الآلة واللغات عالية المستوى، وتحتاج إلى مترجم (Assembler) لترجمتها إلى لغة الآلة التي يفهمها الحاسوب. وهي مرتبطة بالآلة أي بالمعالج (Processor)، حيث لكل معالج لغة تجميعية خاصة فيه.

1-4-1-2- اللغة التجميعية (Assembly language) :

وهي لغة رمزية (Mnemonic) تستخدم الرموز بدل الأرقام الثنائية ولذلك فإن كل سطر في هذه اللغة يُعد ممثلاً لأمر واحد في لغة الآلة وهي أكثر سهولة من لغة الآلة إذ هي أكثر قابلية للحفظ وسهلة في الإعداد والمراجعة والتعديل.

1-4-2- اللغات عالية المستوى (High Level Languages):

بسبب عدم المقدرة على تنفيذ البرنامج المكتوب بلغة الآلة عند نقله إلى حاسوب من طراز آخر ، ولأن إعادة صياغته تحتاج لمجهود كبير ومحصورة ببعض المتخصصين في المعلوماتية، كان من الضروري إيجاد لغة لكتابة البرامج بشكل مستقل عن طراز

الحاسوب، وسميت هذه اللغة باللغة العالية المستوى، وهي لاترتبط بنوع الحاسوب بل تكون قريبة بأسلوب كتابتها من اللغة الطبيعية (اللغة الإنكليزية عموماً) وكل اللغات العالية المستوى تحتاج إلى برنامج آخر لكي يحولها إلى لغة يفهمها الحاسوب وهذا البرنامج هو إما المترجم (Compiler) أو المفسر (Interpreter) .

وتتعدد لغات البرمجة عالية المستوى بحيث تتلاءم كل واحدة منها في مجال تطبيق معين مختلف عن الآخر ومنها على سبيل المثال :

- لغة Basic وتتميز بسهولة التعلم والاستعمال وتصلح للبرامج الصغيرة والمتوسطة، وهي اختصار لـ :

  Beginner's all-purpose symbolic instruction code
- لغة الفورتران Formula Translation وتستخدم في التطبيقات الهندسية والعلمية، وصنعت من قبل شركة IBM عام 1957 ولها عدة إصدارات.
- ولغة الكوبول Common business oriented language (Cobol) وتستخدم في مجال الأعمال التجارية العامة، وتتميز بقدرتها على تنظيم الملفات وتوثيقها ولكنها لاتستطيع برمجة المعادلات الرياضية والمشاكل الاقتصادية المعقدة.
- لغة (Algol) Algorithm language: أي لغة الخوارزميات ، ووضعت عام 1958 لحل المسائل العلمية والعددية.
- لغة APL : ووضعت عام 1962 وهي سهلة التعلم وتستعمل في معالجة الجداول.
- لغة Pascal : نسبة إلى العالم الرياضي الفرنسي باسكال صُممت في أواخر الستينات وهي تجمع مزايا عدة لغات مثل الفورتران وبيسك وألغول أي يمكن استخدامها في المجالات الرياضية والعلمية والهندسية والتجارية والإدارية.
- لغة (PL/1) Program Language : صممتها شركة IBM في منتصف الستينات لتجمع مزايا لغتي الفورتران وكوبول فظهرت النسخة القياسية منها

طبقاً للمقاييس العالمية ANSI في عام 1976 ، وهي لغة تجمع بين الأغراض الهندسية والعلمية وبين الأغراض التجارية ولكنها محدودة الانتشار بسبب كونها خاصة بشركة IBM ولحاجتها إلى مساحات تخزينية كبيرة في ذاكرة الحاسوب.

- لغة أدا ADA : صممت في وزارة الدفاع الأمريكية وسميت باسم أول مبرمجة وهي أجاتا أدا بارون وكان الهدف منها أن تكون لغة قياسية للأنظمة العسكرية، ثم استخدمت بعد ذلك للأغراض التجارية، تساعد هذه اللغة في تصميم المنظومات الهندسية إلاّ أن استخدامها في الحواسيب الصغيرة يكون صعباً لحاجتها إلى سعة ذاكرة كبيرة.

1-5- تمثيل المعلومات في الحاسوب:

يعتمد تمثيل المعلومات في الحاسوب على طبيعة المعلومات الممثلة، فيختلف مثلاً تمثيل المعلومات الحرفية عنها في العددية، وكذلك تمثل الأعداد الصحيحة بطريقة مختلفة عن طريقة تمثيل الأعداد الحقيقية، وبرغم هذا الاختلاف في تمثيل المعلومات فإنها جميعاً تعود إلى نظام تمثيل أساسي وهو التمثيل الثنائي (Binary System).

1-5-1- نظام الترميز الثنائي أو الترميز الرقمي (Binary Digit):

العنصر الأساسي للتمثيل الثنائي هو الرقم الثنائي (Binary Digit) أو البت (bit) ويسمح هذا النظام بتمثيل معلومتين مختلفتين بوساطة مقدار فيزيائي بأخذ إحدى حالتين ممكنتين فقط، كأن نستخدم مثلاً المصباح الكهربائي بحيث يكون إما مضاءاً أو مطفأً ، أو المكثفة الكهربائية فهي تكون إما مشحونة أو فارغة الشحنة. إذن يمكن تمثيل الرقم الثنائي بمربع يمكن أن يكون في إحدى حالتين فقط أبيض أو أسود.

لنفترض أننا نريد تمثيل معلومة مثل اللون، بحيث يمكن أن نأخذ مجموعة من القيم مثلاً : { أبيض ، أسود ، أحمر ، أزرق، أخضر، أصفر ، بني } .

كما وضحنا سابقاً البت الواحد يمكن أن يأخذ إحدى القيمتين صفر أو واحد باستخدام 2 بت حيث يمكن تمثيل 2^2 حالة مختلفة ومنه لتمثيل قيم الألوان السابقة تحتاج لثلاثة بتات تأخذ كل منها القيم 0 أو 1 كما يلي:

أبيض	000	أخضر	100
أسود	001	أصفر	101
أحمر	010	بني	110
أزرق	011		

يمكن الاستنتاج أنه يمكن تمثيل 2^N حالة مختلفة للمعلومة الواحدة باستخدام مجموعة مؤلفة من N رقماً ثنائياً. في المثال السابق باستخدام $N = 3$ يمكن تمثيل $2^N = 2^3 = 8$ حالة مختلفة، وآخرها هو الحالة التي تأخذ كل البتات القيمة 1 ولنفرض أنها ترمز للون الرمادي.

1-5-2- نظام الترميز الثماني (Octal System):

يسمح هذا النظام بتمثيل ثماني حالات مختلفة تبدأ بالقيمة العددية صفر وتنتهي بالرقم 7. إذن كل الأرقام من 0 ← 7 هي أرقام مقبولة في نظام الترميز الثماني، حيث هذا النظام أساسه هو العدد 8.

يمتاز النظام الثماني بسهولة تحويل الأعداد منه إلى النظام الثنائي، إذ يكفي استبدال كل رقم ثماني من العدد بثلاثة أرقام ثنائية، وعند تحويل الأعداد الثنائية إلى ثُمانية تقسم الأرقام الثنائية إلى مجموعة مكونة من ثلاثة أرقام، ثم تستبدل كل مجموعة بالرقم الثماني المقابل لها، مثال:

$$1247\big|_8 = \underset{1}{001}\quad \underset{2}{010}\quad \underset{4}{100}\quad \underset{7}{111}\Big|_2$$

$$101\quad 011\quad 100\quad 011\quad 101\big|_2 = 53435\big|_8$$

1-5-3- نظام الترميز الست عشري (Hexadecimal System):

أساس هذا النظام هو العدد 16 ، ومن اسمه يسمح هذا النظام بترميز ست عشرة حالة مختلفة وهي القيم 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 ثم الأحرف F,E,D,C,B,A لتمثيل القيم من 10 حتى 15 وبذا يكون المجموع 16 رقماً مختلفاً. كما ذكرنا في نظام الترميز الثنائي أنه لتمثيل 2^N حالة نحتاج إلى N بت ، ولتمثيل كل رقم ثماني نحتاج إلى

3 أرقام ثنائية (3 bits)، كذلك نحتاج لتمثيل كل رقم ست عشري إلى 4 أرقام ثنائية (4 bits). وبالتالي يمكن التعبير عن العدد الثنائي:

$$10\ 101\ 101|_2 = 255|_8 = AD|_{16}$$

1-5-4- نظام الترميز الثنائي العشري B.C.D. (Binary Coded Decimal):

تتكون الأرقام في نظام الترميز الثنائي العشري من عشرة أرقام واقعة ضمن المجال من 0 ← 9 ويمثل كل منها داخل الحاسوب بأربعة أرقام ثنائية.

مثال:

$$0111\ 0101\ 0010\ 1001\,|_2 = 7529|_{10}$$

طبعاً الأحرف من A ← F غير مستخدمة لأننا نمثل بالنظام العشري وهي ليست بأرقام عشرية.

1-6- التحويل من نظام الترميز العشري إلى نظام الترميز الثنائي:

إن نظام الترميز المستخدم في حياتنا اليومية هو نظام الترميز العشري (Decimal system) والذي أساسه هو العدد 10 ، وإن قسمة أي عدد عشري على أساس النظام (العدد 10) قسمة صحيحة سينتج رقماً يمثل ناتج القسمة ورقماً آخر يمثل باقي القسمة. فمثلاً بقسمة العدد 36 على الأساس 10 قسمات صحيحة سينتج الرقم 3 وهو الناتج الصحيح والباقي هو الرقم 6 ، إذ في الواقع الرقم 6 هو خانة الآحاد من العدد 36 في نظام الترميز العشري. بقسمة العدد الناتج 3 على الأساس 10 من جديد يكون لدينا الرقم 3 كباقي للقسمة وهو يمثل خانة العشرات من العدد 36 ، وهكذا يمكننا الحصول على خانات أي عدد في نظام الترميز العشري.

وبهذه الطريقة يمكننا تحويل أي عدد بنظام الترميز العشري إلى أي نظام ترميز آخر بالقسمة المتتالية للعدد العشري على أساس النظام المراد تحويله إليه.

وسلسلة الأرقام الممثلة لباقي القسمة في كل مرحلة هي سلسلة عددية تمثل مكافئ العدد بالنظام الجديد. الشكل (1-2) يبين عملية التحويل من النظام العشري إلى النظام الثنائي.

مثلاً لتحويل العدد 36 من نظام الترميز العشري إلى نظام الترميز الثنائي نقسم باستمرار العدد الصحيح الناتج عن القسمة على أساس النظام (العدد 2) إلى أن يصبح ناتج القسمة الصحيحة مساوياً للصفر، عندها يكون العدد الثنائي المكافئ هو سلسلة الأرقام التي هي باقي القسمة . وآخر رقم ناتج عن باقي القسمة يمثل الخانة ذات القيمة العظمى من العدد الثنائي كما هو مبين فيما يلي :

أساس النظام المراد التحويل إليه 2	36 العدد بنظام الترميز العشري
باقي القسمة ← 0	18 → ناتج القسمة الصحيحة
0	9
1	4
0	2
0	1
1	0

$36|_{10} = 100100|_2$

$36|_{10} = 100100|_2$

باقي القسمة	ناتج القسمة الصحيح	القسمة	الخطوة
1	28	57/2	1
0	14	28/2	2
0	7	14/2	3
1	3	7/2	4
1	1	3/2	5
1	0	1/2	6

$111001_2 = 57_{10}$

إذن :

$57|_{10} = 111001|_2$

الشكل (1-2) التحويل من النظام العشري إلى الثنائي

1-7- التحويل من نظام الترميز الثنائي إلى نظام الترميز العشري:

يمكن حساب العدد العشري من مجموع جداءات الأرقام المكونة للعدد العشري والتي تأخذ القيم من 0 , 1 .. حتى 9 ، بقيم الخانات الموجودة فيها الأرقام.

مثال:

مكافئ العدد الثنائي $1011101|_2$ يحسب بالشكل:

$$1\times 2^0 + 0\times 2^1 + 1\times 2^2 + 1\times 2^3 + 1\times 2^4 + 0\times 2^5 + 1\times 2^6 =$$
$$1 + 0 + 4 + 8 + 16 + 0 + 64 = 93|_{10}$$

علماً أن البت الأيمن هو البت ذو القيمة الدنيا والبت الأيسر هو البت ذو القيمة العليا.

1-8- المراحل المطلوبة لوضع برنامج للحاسوب:

يجب أن يكون واضحاً لدينا أنه ليس باستطاعتنا بمجرد حصولنا على الحاسوب – صغيراً كان أم كبيراً – ودراسة طريقة تشغيله أن نحل أية مشكلة نريدها. فالحاسوب ليس مثل بقية الأجهزة التي اعتدنا استخدامها، بالرغم من كونه يستطيع تنفيذ مسائل دقيقة ومعقدة ويقدم دراسات تحليلية واقتصادية ومنطقية، إلاّ أن كل ذلك يكون بفضل برامج قد يستغرق إعدادها شهوراً أو سنوات لكي نضمن أن الحاسوب سيقوم بالتنفيذ على أكمل وجه وبمختلف الحالات الخاصة والعامة التي يمكن أن تعترضه في أثناء تنفيذ الحل. إذن قبل أن يأتي دور الحاسوب في معالجة المعلومات واستخلاص النتائج علينا تلقينه برنامجاً لكيفية الحل ويجب أن يكون هذا البرنامج معداً وفق منطق خاص وبأوامر يتقبلها الحاسوب وبشكل نضمن معه أنه سيراقب العمليات التي ستتم في أثناء تنفيذ البرنامج.

لوضع برنامج ما ، علينا بشكل عام القيام بالمراحل التالية:

1- فهم وتعريف المسألة بشكل صريح من ناحية المعطيات التي تسمى مُدخلات (Input) والتي ستغذي الحاسوب وفق البرنامج. بعض المسائل يكون تحديد معطياتها سهل بينما بعضها الآخر يكون صعباً ويحتاج لدراسة تحدد حجم المعطيات، وهل من المفيد زيادة عدد هذه المعطيات: حجمها، نوعيتها، الترتيب الذي يجب أن تعطى فيه للحاسوب. وتتم دراسة ومناقشة هذه ا لمعطيات مع مختصين سيستثمرون البرنامج وبذلك نضمن فعاليته وفائدته.

2- تحديد الغاية المرجوة من البرنامج أي النتائج اللازم الحصول عليها، أو ما هو مطلوب تعيينه أو حسابه كذلك تحديد حجمه ونوعيته والترتيب المطلوب إخراجه فيه، ومن المفضل أيضاً مناقشة ذلك مع مختصين. تسمى النتائج بالمخرجات (Output).

3- وضع خطة معينة ومحددة للحل تتضمن القوانين والعلاقات والمعادلات والطرق الحسابية والمنطقية التي نتوصل بفضلها إلى النتائج باستخدام المعطيات.

وبشكل عام لكل مسألة أكثر من طريقة حل نستطيع باعتمادها التوصل إلى النتائج المطلوبة. ويمكن اعتماد إحدى الطرق من دون الأخرى إذا كان بالإمكان تحويلها بسهولة إلى ألغوريتم (Algorithm) دقيق ومفصل ومبرمج ويحتاج إلى عدد أقل من التعليمات للوصول إلى النتائج، ويجب علينا هنا أن نناقش أيضاً هل سيتم دوماً تنفيذ هذه الطريقة بسرعة مهما اختلفت المعطيات.

4- تشمل هذه المرحلة تحويل الطريقة المعتمدة إلى ألغوريتم مبرمج يتقبله الحاسوب ثم يتم بعد ذلك تلقينه للحاسوب. هذه المرحلة تُعد بالنسبة للمبرمج المبتدئ من أصعب المراحل. وفيها يتوجب عليه أن يتذكر دائماً:

1) أن الحاسوب ينفذ البرنامج دائماً من دون تدخل من طرفنا، لأن كل عمليات الحساب والمحاكمة تتم من قبله وفق خطوات البرنامج، ويمكننا التدخل وتصحيح الأخطاء بعد انتهائه من التنفيذ واختلاف نتائجه مع النتائج المتوقعة. أما في أثناء

التنفيذ فالحاسوب يقوم بمراقبة العمل والنتائج الانتقالية ولايمكننا التدخل خلالها. هذا يعني ضرورة توضيح ودراسة جميع الحالات التي يتوقع وصول الحاسوب إليها عند تنفيذه للبرنامج وكيفية التعرف على كل حالة. فمثلاً لو تطلب الحل إجراء عمليات خاصة عندما تأخذ النتيجة الانتقالية قيمة ما تبعاً لمعطيات خاصة فعلينا مسبقاً بالبرنامج ذكر ما هي العمليات الواجب إجراؤها في مثل هذه ا لحالة.

إذا أردنا من الحاسوب التوقف عن متابعة التنفيذ إذا حصل على نتيجة كبيرة بسبب معطيات خاطئة فعلينا أيضاً أن نبين ذلك في البرنامج وتبيان القيمة التي إذا بلغتها النتيجة فعلى البرنامج التوقف عن متابعة التنفيذ ، مثلاً طبع جملة تخبرنا أن المعطيات خاطئة أو القيمة أصبحت كبيرة أو غير ذلك، ثم التوقف عن التنفيذ أو الانتقال إلى بداية البرنامج مثلاً.

بالتالي الحاسوب لايستطيع القيام بالمحاكمة بنفسه وعلينا تلقينه الخطوات الواجب عليه تنفيذها في كل حالة متوقعة ضمن الألغوريتم.

2) إن الحاسوب لايمكنه أن يخمن أو يتحسس لشيء، فمثلاً : لو كانت معطيات المسألة من طبيعة أو نوع مختلف (أرقام وأحرف) ولو كان من ضمن خطوات البرنامج أمر قيام بعملية جمع لهذه المعطيات، فهو لن يعرف الفرق وسيحاول القيام بإجراء عملية الجمع من دون تردد، بينما الإنسان وبمجرد النظر إلى المعطيات يستطيع تحديد عدم إمكانية إتمام عملية الجمع.

مثال آخر : لو كانت المعطيات عبارة عن أطوال بعضها معطى بالأمتار وبعضها بالسنتيمتر فإن كان المطلوب جمع هذه الأطوال فإن الحاسوب سيجمعها من دون تحويلها إلى الواحدة نفسها وبالتالي يكون ناتج الجمع خاطئاً.

مثال آخر : إذا كان المطلوب جمع قائمة من الأعداد فإن الإنسان يتوقف عن الجمع بانتهاء القائمة مع أنه قد لا يعلم عدد الأعداد في القائمة، ولكن الحاسوب لايمكنه القيام

بعملية الجمع أصلاً لهذه القائمة ما لم يكن يعرف مسبقاً طول هذه القائمة، أو كان يميز هذه النهاية بطريقة ما كوضع قيمة معينة تدل على نهاية القائمة.

إذن علينا دوماً تحديد المطلوب من الحاسوب عند كل حالة من الممكن أن يصادفها خلال تنفيذه للبرنامج مهما كانت بسيطة وبديهية ذلك لأنه كما قلنا هو غير قادر على إجراء المحاكمة لوحده.

الفصل الثاني

الخوارزميات ومخططات سير العمليات

Algorithms and Flow Charts

2-1- مقدمة:

الخوارزمية مصطلح قديم يعود لمطلع القرن التاسع الميلادي في أوج الدولة العباسية في زمن المأمون. ولكن ازداد في الفترة الأخيرة عند ظهور الحواسيب الاهتمام بهذا المصطلح وشاع استخدامه في الكتب والأبحاث وميادين متعددة من النشاطات العلمية والتطبيقية.

والخوارزمية هي نسبة إلى العالم العربي محمد بن موسى الخوارزمي الذي برز في علوم الرياضيات والفلك وهو أول من وضع مبادئ علم الجبر.

واليوم يرتبط مصطلح الخوارزمية ببرمجة الحواسيب الإلكترونية وتُعرف بأنها مجموعة الخطوات المتسلسلة والمحددة التي تؤدي إلى الوصول إلى حل لقضية والوصول إلى النتائج ، أي هي متتالية من العمليات المعرفة والمنتهية اللازمة لإنجاز عمل أو حل مسألة والحصول على نتيجة صحيحة. والخوارزمية تعالج المعطيات المدخلة وغالباً ما يجب أن تتضمن عمليات تأكد من صحة المعطيات. حيث المعطيات المدخلة لاتقتصر على الأعداد والأرقام بل تشمل الرموز والنصوص والرسوم والصور والأصوات كمعطيات دخل ونتائج خرج، أي يمكننا أن نتحدث مثلاً عن خوارزمية ترتيب مجموعة أسماء أبجدياً، أو خوارزمية تعرف جملة منطوقة أو خوارزمية تعرف شكلاً مرسوماً.

عادة نقوم بوضع الخوارزمية اللازمة قبل كتابة البرنامج وتلقينه للحاسوب لتنفيذه. وكما ذكرنا يمكن وجود أكثر من خوارزمية لكل مسألة ولكن أفضل

الخوارزميات هي التي تصل إلى النتيجة بأقل عدد من التعليمات والعمليات وأقل زمن وجهد ممكن والتي تكون سهلة الفهم وسريعة التنفيذ.

2-2- أنواع الخوارزميات:

يمكن إعادة تعريف الخوارزمية بأنها توصيف دقيق وكامل في خطوات متسلسلة ومعدودة تحدد طريقة إنجاز عمل ما أو حل مسألة ما ويمكن تقسيم الخوارزميات إلى نوعين حسابية وغير حسابية.

2-2-1- الخوارزمية غير الحسابية:

من أكثر الخوارزميات غير الحسابية المستخدمة في مجال الحاسوب هي خوارزمية معالجة النصوص، وتخزين المعلومات واستعادتها، وإدارة قواعد البيانات ويمكننا أن نقول مثلاً: خوارزمية كيفية طرح سؤال ضمن محاضرة، أي كيفية الاستفسار عن قضية معينة ضمن المحاضرة تتلخص بالخطوات التالية:

1- الانتظار حتى وصول المحاضر إلى نهاية المقطع الكلامي.

2- رفع اليد لأخذ الأذن بالكلام.

3- خفض اليد والمباشرة بطرح السؤال.

4- الاستماع للإجابة حتى النهاية وفي حال عدم شموليتها للسؤال تعاد الخطوات من 1 حتى 4 .

2-2-2- الخوارزمية الحسابية:

وهي الخوارزمية التي تتعامل مع المقادير الرياضية، مثلاً:

$$z = \frac{x!}{2x-4}$$

يمكن حل هذه ا لخوارزمية بمعرفة العمليات الحسابية وكيفية إجرائها وقيمة x. الصيغة الرياضية السابقة التي تحسب المقدار z هي طبعاً ليست خوارزمية بالرغم من أن هذه الصيغة واضحة وتبين العمليات اللازمة لحساب المقدار z إلاّ أنه لايعطي تسلسل

هذه العمليات، فمن الممكن أن نبدأ بحساب البسط ثم المقام ثم نقسم البسط على المقام للوصول إلى الجواب، كما يمكن البدء بشكل معاكس. ولكي تصبح أي صيغة رياضية خوارزمية لابد أن تقترن بتسلسل تنفيذ عملياته أي إضافة بعض الشروط والقواعد، كأن نبدأ دوماً وفق أفضليات تحدد بحسب نوع المسألة المطروحة وجعل هذه الأفضليات قواعد للتنفيذ تمكننا من الوصول إلى خوارزمية صالحة للتنفيذ.

2-3- طرق كتابة الخوارزمية:

يمكن التعبير عن الخوارزمية بعدة طرائق تختلف من حيث دقة التعبير وسهولة الفهم وأهم الطرائق المستخدمة في كتابة الخوارزميات هي:

1) صياغتها باللغة الطبيعية أي اللغة المتداولة كاللغة العربية أو اللغة الإنكليزية وهي الطريقة التلقائية لصياغة الخوارزميات ونستخدمها يومياً تقريباً.

2) صياغتها بطريقة بيانية بوساطة أساليب عديدة نذكر منها المخطط النهجي أومخطط سير العمل.

3) صياغتها باستخدام لغة رمزية خاصة.

ولقد جرت العادة على استخدام أكثر من طريقة في أثناء مرحلة إنشاء الخوارزمية مثل استخدام المخططات النهجية والتعبير عن الخطوات باللغة الطبيعية أو الرمزية ضمن الأشكال والرموز الاصطلاحية الخاصة بهذه المخططات كما سنرى لاحقاً.

2-3-1- استخدام اللغة الطبيعية في صياغة الخوارزمية:

تستخدم اللغة الطبيعية في صياغة بعض أنواع الخوارزميات الموجهة إلى الجمهور كتعليمات تشغيل الأجهزة واستخدامها أو وصفات تحضير أطباق الطعام، مثال على خوارزمية مصاغة باللغة الطبيعية هي خوارزمية طرح سؤال ضمن محاضرة سابق الذكر، وفيما يلي خوارزمية عبور شارع بأمان:

1- انتظار ظهور لون الإشارة الأخضر لعبور المشاة.

2- التحرك لقطع النصف الأول من الشارع ناظرين إلى جهة قدوم السيارات.

3- التوقف عند منتصف الشارع للتأكد من خلو النصف الثاني من السيارات.

4- قطع النصف الثاني من الشارع ناظرين إلى جهة قدوم السيارات.

2-3-2- استخدام الطريقة الرمزية في صياغة الخوارزمية:

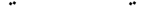
من أهم طرائق ترميز الخوارزمية رمزياً استخدام لغات البرمجة المختلفة والتي سندرس منها لغة C++.

2-3-3- استخدام الطريقة البيانية في صياغة الخوارزمية:

تُستخدم في هذه الطريقة أشكالٌ هندسية خاصة توضح خطوات تنفيذ الخوارزمية وأسهم تصل بينها إضافة إلى عبارات باللغة الطبيعية أو بتعابير رياضية أو منطقية وبذلك تعطي وصفاً تصويرياً لخطوات الخوارزمية فتسهل تتبع التسلسل المنطقي لحل المسألة، وتُعد مخططات سير العمليات (Flow chart) أكثر الأشكال انتشاراً واستخداماً في توصيف الخوارزميات حيث تحدد طريقة تسلسل وترابط خطوات تنفيذ الخوارزمية عن طريق الربط بين رموز اصطلاحية تمثل تتالي عمليات تشير إلى البداية، النهاية، الإدخال والمعالجة، وإخراج المعطيات والنتائج، ويمكننا بوساطة هذه المخططات تحديد العلاقة المنطقية بين كافة خطوات الحل ومواقعها ووظائفها.

نبين فيما يلي أهم الرموز الاصطلاحية التي اعتمدت، حيث يمثل كل شكل إحدى الفعاليات الواجب إنجازها، وهي معتمدة من قبل المعهد القياسي الأمريكي ANSI: American National Standard Institute .

1- رمز البداية أو النهاية Start/end symbol :

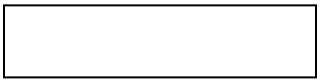


يستخدم هذا الرمز في بداية ونهاية المخطط ونكتب في داخله كلمة ابدأ Start عند البدء، وكلمة نهاية End أو Stop عن النهاية.

2- رمز الإدخال أو الإخراج Input/output Symbol:

يستخدم لتحديد المدخلات بكتابة Input ثم قائمة المدخلات في الدخل أو كتابة Output ثم قائمة المخرجات في الخرج.

3- رمز المعالجة Processing symbol :

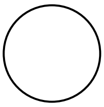


يستخدم لكتابة العمليات الحسابية أي التعيين، وهي العملية اللازمة للحصول على النتائج .

4- خطوط اتجاه سير العمل Flow lines :   ←   →   ↓   ↑

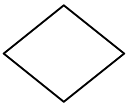
تستخدم لوصل رموز المخطط مع بعضها واتجاه هذه الخطوط يحدد اتجاه سير العمل .

5- رمز الاتصال Connector symbol :



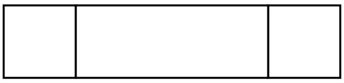
يستخدم لربط أجزاء من المخطط عندما يكون المخطط كبيراً أو موزعاً على عدة صفحات.

6- رمز اتخاذ القرار Decision symbol :



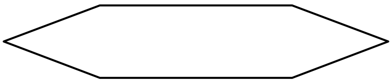
يستخدم للمقارنات الحسابية أو المنطقية وتكون النتيجة إما نعم Yes أو لا No للمقارنات الحسابية و False خطأ أو True صح للمقارنات المنطقية.

7- رمز البرامج الجزئية Subroutine symbol :



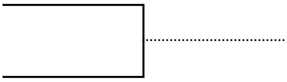
ويستخدم لاستدعاء برنامج فرعي وإدخاله ضمن البرنامج الرئيسي.

8- رمز التكرار Loop symbol :



يستخدم لتكرار جزء من العمل عدداً من المرات .

9- رمز الملاحظات Comment symbol :



يستخدم للملاحظات الإضافية، والخط المنقط يكون باتجاه الرمز المحتاج إلى الملاحظة.

2-4- أنواع مخططات سير العمليات :

يمكن أن نميز ثلاثة أنواع من المخططات وهي :

1- مخططات تتابعية أو تسلسلية Sequential flow chart

2- مخططات تفرعية Branched flow chart .

3- مخططات تكرارية repetitive flow chart

2-4-1- مخططات تتابعية (تسلسلية) :

تستخدم في المسائل التي يتطلب حلها خطوات محددة متتالية تؤدي إلى النتيجة من دون الحاجة إلى تغيير سياق التنفيذ . وتنفذ هذه الخطوات خطوة خطوة حتى الوصول إلى النهاية من دون الحاجة إلى تجاهل أو تكرار لأي من الخطوات الموجودة. ويقدم المثال التالي أحد هذه المخططات.

مثال:

المطلوب رسم مخطط سير العمليات لخوارزمية إيجاد مساحة ومحيط مستطيل عُرف ضلعاه هما A و B .

يمكن كتابة الخوارزمية باللغة الطبيعية بالشكل التالي:

1- ابدأ

2- ادخل طول الضلع الأول A .

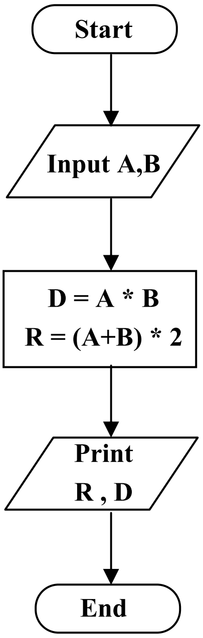
3- ادخل طول الضلع الثاني B .

4- احسب : المساحة = طول الضلع الأول × طول الضلع الثاني

5- احسب : المحيط = (طول الضلع الأول + طول الضلع الثاني) * 2

6- اطبع قيمة المساحة والمحيط.

7- نهاية.



الشكل (2-1) مخطط سير العمل

لخوارزمية حساب محيط ومساحة مستطيل عُلم طول ضلعاه

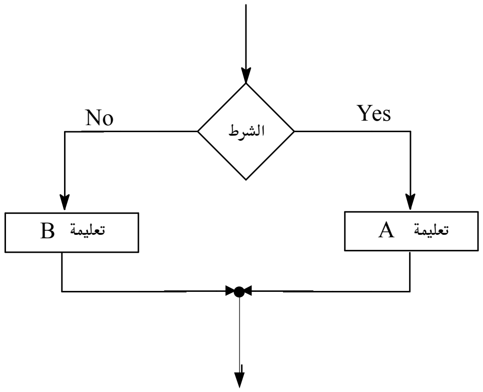
2-4-2- مخططات تفرعية :

تستخدم في المسائل التي تخضع لشروط تحدد التتالي المناسب للخطوات المطلوب تنفيذها. حيث يفرض تحقق الشرط أو عدمه، الاختيار بين طريقين أو عدة طرق .

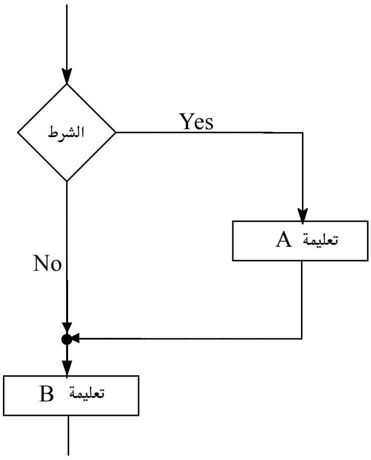
المخطط (a,2-2) يوضح أنه إذا كان الشرط المختبر محققاً والجواب Yes عندها تنفذ مجموعة التعليمات A ثم يتابع تنفيذ بقية تعليمات البرنامج، أما إذا كان الشرط غير محقق والجواب No عندها تنفذ مجموعة التعليمات B ثم بقية تعليمات البرنامج.

المخطط (b-2-2) يوضح أنه إذا كان الشرط المختبر محققاً والجواب Yes عندها تنفذ مجموعة التعليمات A ثم مجموعة التعليمات B وباقي تعليمات البرنامج.

أما إذا كان جواب الشرط No عندها تنفذ مجموعة التعليمات B فقط ثم يتابع تنفيذ بقية تعليمات البرنامج.



الشكل (a-2-2)



الشكل (b-2-2)

مثال : ارسم مخطط سير العمليات للخوارزمية التي تمثل حل المعادلة التالية من أجل قيمة وحيدة لـ y ، مَثِّل هذه الخوارزمية باللغة الطبيعية أيضاً.

المعادلة معطاة بالشكل:

$$z = \frac{5y + 3}{3y - 5}$$

تمثيل الخوارزمية باللغة الطبيعية:

1- ابدأ.

2- ادخل قيمة y.

3- احسب $A = 5y + 3$

احسب قيمة $B = 3y - 5$

4- إذا كانت قيمة $B = 0$

اذهب إلى الخطوة "7"

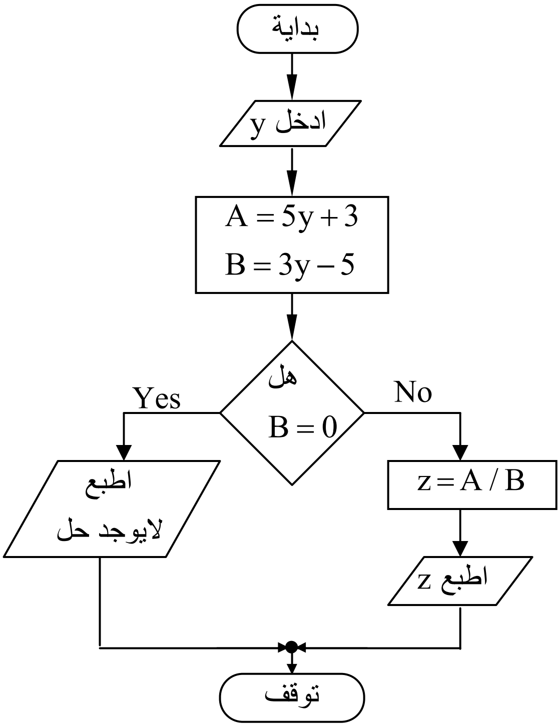
5- احسب $z = A/B$

6- اطبع قيمة z واذهب إلى الخطوة "8".

7- اطبع العبارة " لايوجد حل".

8- توقف.

كما تمثل الخوارزمية بمخطط سير العمل كما في الشكل (2-3)



الشكل (2-3) **مخطط سير العمل لخوارزمية تفرعية**

2-4-3- مخططات تكرارية:

تستخدم في المسائل التي يتطلب حلها تكرار مرحلة واحدة أو أكثر لعدة مرات. حيث يحدد الشرط الذي يقرر عدد مرات التكرار بوساطة صندوق الاختبار المرتبط بزيادة أو إنقاص متحول يسعى إلى تحقيق الشرط المحدد ضمن صندوق الاختبار.

نحتاج في كثير من تطبيقاتنا العملية إلى أن نقوم بتكرار تنفيذ مجموعة من التعليمات. مثلاً حساب عدد الطلاب الناجحين، أو جمع مجموعة من القيم تمثل ظاهرة ما كحساب معدل نجاح الطالب.

نعتمد في الحالة الأولى على خوارزمية العد (Counting algorithm) التي تتمثل بالمراحل التالية :

1- إعطاء قيمة أولية للعداد تساوي الصفر : Counter = 0

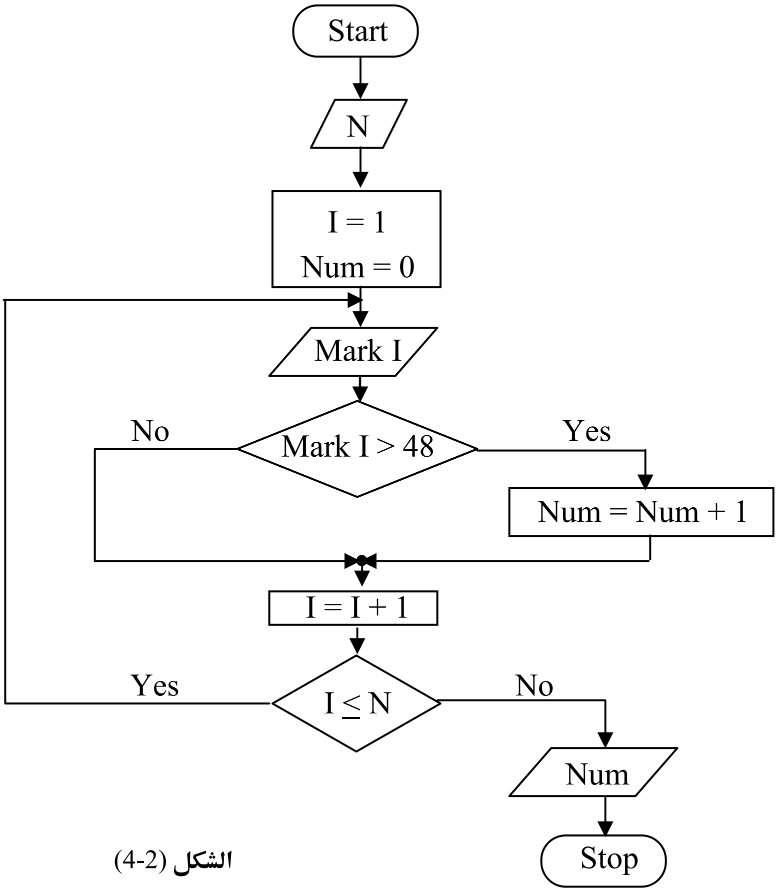
2- إعطاء قيمة جديدة للعداد تساوي القيمة الأولية زائد واحد :

Counter = Counter + 1

3- تنفيذ العمليات المطلوبة عدداً من المرات.

4- إعادة تنفيذ الخطوات 2 و 3 حسب المسألة المطروحة.

الشكل(2-4) التالي يبين مخطط سير العمليات الذي يقوم بحساب عدد الطلاب الناجحين من N طالب في مادة البرمجة.



الشكل (2-4)

أما في الحالة الثانية فنعتمد على خوارزمية التجميع (Summing Algorithm)، حيث نحتاج إلى متغيرين أحدهما نضع فيه القيمة المراد جمعها والثاني نضع فيه حاصل

جمع القيم ويمكن تمثيل مراحل هذه الخوارزمية كما يلي:

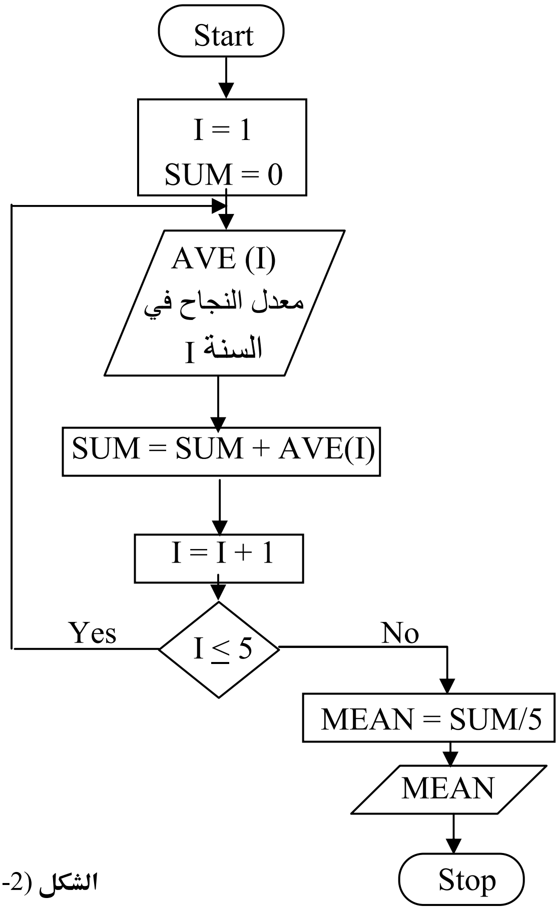
1- إعطاء قيمة تساوي الصفر لمتغير الجمع أو أية قيمة أخرى مناسبة حسب المسألة المدروسة : SUM = 0 .

2- قراءة القيمة الأولى المراد جمعها ولتكن y .

3- إضافة القيمة المقروءة إلى قيمة متغير الجمع : SUM = SUM + y

4- إعادة تنفيذ الخطوات 2 و 3 حتى تنتهي من جمع كل القيم.

الشكل (2-5) يبين مخطط سير العمليات الذي يمثل خوارزمية إيجاد معدل التخرج لطالب في كلية الهندسة.



الشكل (2-5)

مثال : ارسم مخطط سير العمليات لخوارزمية إدخال علامة ثلاث مواد لعدد من الطلاب، ثم طباعتها وطباعة المعدل. إذا فرضنا أن عدد الطلاب هو عشرة، وعلامة كل طالب في المواد الثلاث مخزنة في المتحولات C,B,A. مثل هذه الخوارزمية باللغة الطبيعية أيضاً.

تمثيل الخوارزمية باللغة الطبيعية:

1- ابدأ

2- كرر الخطوات من 3 حتى 5 عشر مرات ثم اذهب إلى 6

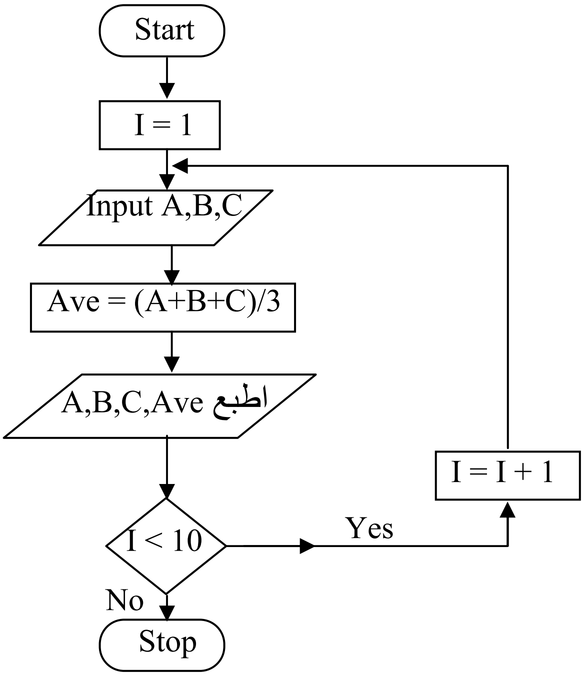
3- ادخل C,B,A .

4- احسب Ave=(A+B+C)/3

5- اطبع A,B,C, Ave

6- توقف.

الشكل (2-6) يبين مخطط سير العمليات خوارزمية المثال السابق.



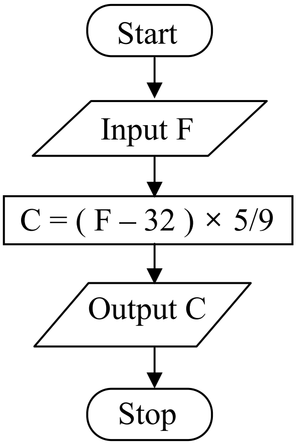
الشكل (2-6)

إن تمثيل الخوارزمية بوساطة مخطط سير العمليات، يعطي صورة واضحة ومتكاملة للخطوات المطلوبة للوصول إلى النتيجة، ويسهل تحديد الخطأ ونوعه إن وجد، وطريقة معالجته. كما تفيد في متابعة حالات المسائل المعقدة ذات التفرعات والتكرارات.

كما أن تمثيل الخوارزمية عن طريق مخطط سير العمليات هو أكثر دقة ووضوحاً من صياغتها باللغة الطبيعية وخاصة إذا كانت معقدة.

2-5- تمارين:

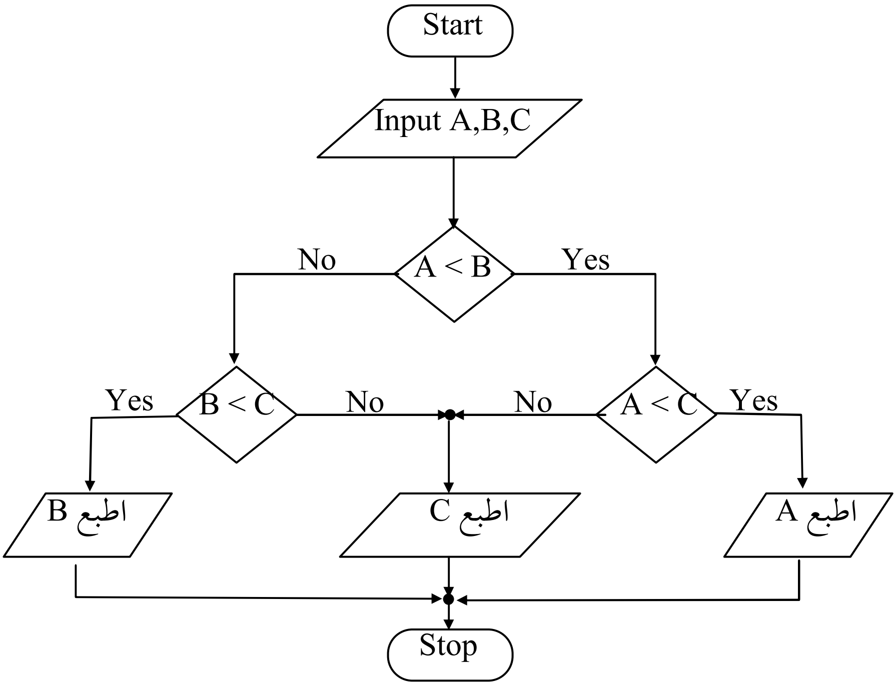
1- ارسم مخطط سير العمليات لخوارزمية تحويل درجة الحرارة من الفهرنهايت F إلى الدرجة المئوية C .(الدرجة المدخلة بالفهرنهايت والدرجة المخرجة بالدرجة المئوية).



الشكل (2-7)

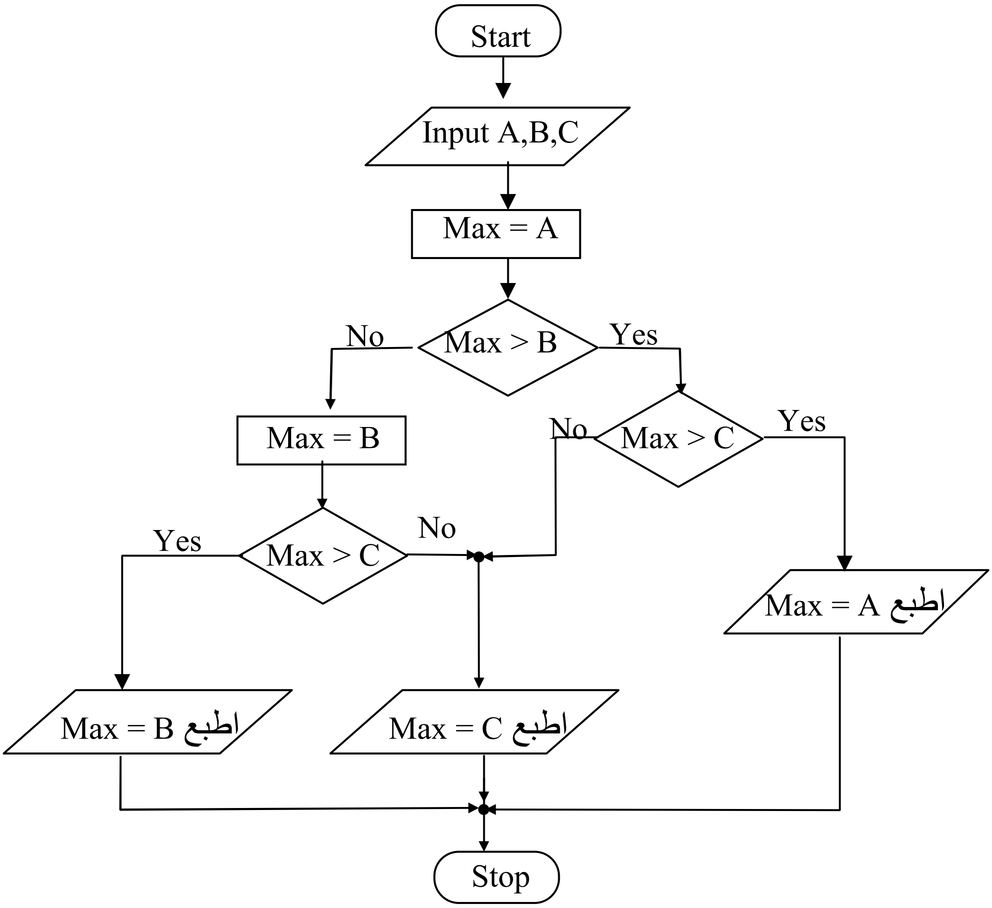
2- ارسم مخطط سير العمليات لخوارزمية إيجاد القيمة الصغرى بين ثلاث قيم مدخلة.

حيث الدخل هو ثلاث قيم A,B,C والخرج هو القيمة الصغرى منهم.



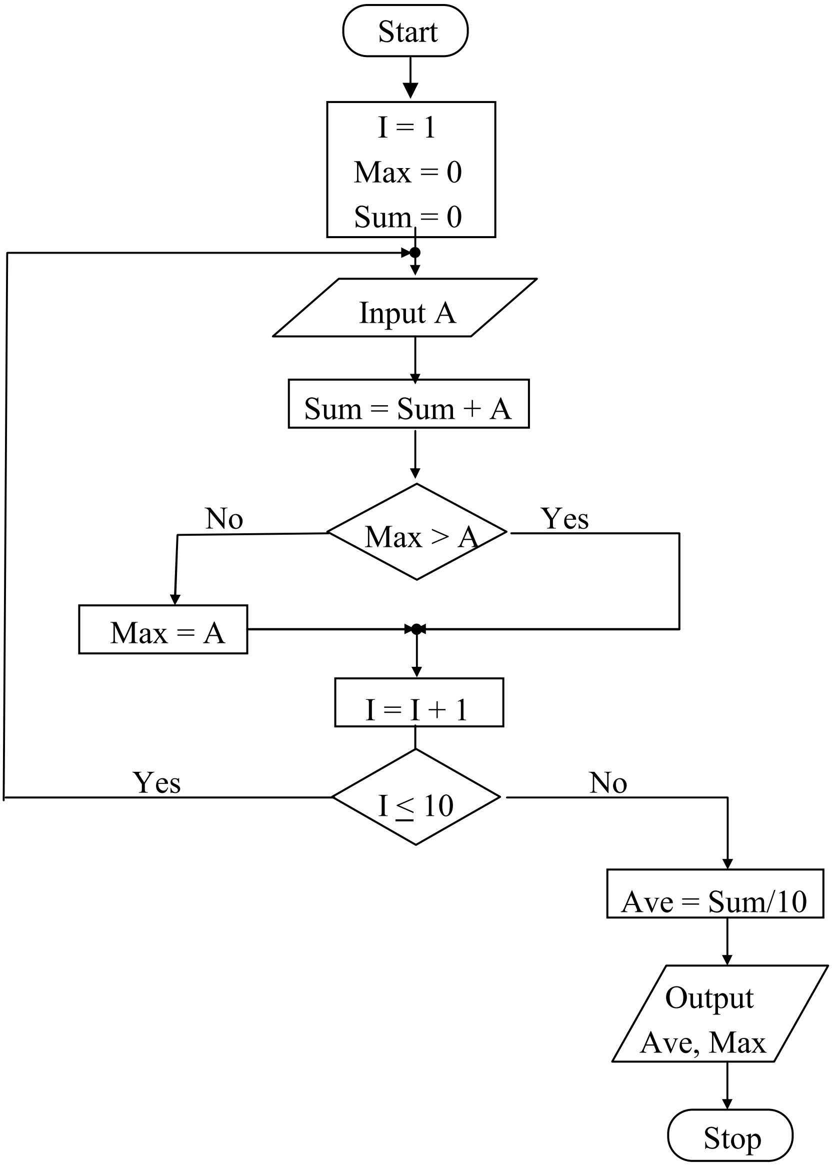
الشكل (2-8)

3- أعد حل التمرين السابق ولكن لإيجاد القيمة العظمى من بين القيم الثلاث المدخلة.



الشكل (9-2)

4- ارسم مخطط سير العمليات لخوارزمية إيجاد متوسط 10 قيم مدخلة وطباعته وكذلك طباعة القيمة العظمى من بين تلك القيم.



الشكل (10-2)

5- ارسم مخطط سير العمليات لخوارزمية إيجاد أصغر عدد من مجموعة أعداد وترتيب هذا العدد في هذه المجموعة.

6- ارسم مخطط سير العمليات لخوارزمية حساب التابع.

$$y = \frac{a + b\cos x}{ax^2 - b\sin^3 x}$$

حيث b, a ثوابت. احسب واطبع قيم التابع y من أجل قيم x التي تتغير ما بين x_0 , x_n وبتزايد قدره ℓ . اعتبر أن $x_0 < x_n$.

7- ارسم مخطط سير العمليات لخوارزمية حساب مجموع الأعداد الموجبة ومجموع الأعداد السالبة في مجموعة من N عدد مدخل.

8- ارسم مخطط سير العمليات لخوارزمية حساب العلاقة :

$$C = (A + B)^2$$

حيث : $A = x + y$ و $B = A + x * y$

وطباعة كل من x و y و A و B و C

9- ارسم مخطط سير العمليات لخوارزمية حساب قيمة التابع التالي :

$$F(x) = \begin{cases} x + 1 & \text{if} \quad x \geq 0 \\ -x + 5 & \text{if} \quad x < 0 \end{cases}$$

10- ارسم مخطط سير العمليات لخوارزمية حساب عاملي عدد حسب ما يلي:

$$N! = \begin{cases} 1 & \text{if} \quad N = 0 \\ 1 * 2 * \ldots * N & \text{if} \quad N > 0 \end{cases}$$

الفصل الثالث

مفاهيم أساسية في لغة البرمجة ++C

Basic concepts of C++ Language

3-1- نبذة تاريخية عن لغة ++C:

تم تطوير لغة ++C انطلاقاً من لغة C والتي بدورها تطورت من لغتين هما BCPL و B . تم تطوير لغة البرمجة C انطلاقاً من اللغة B بوساطة Bennis Ritche في مخابر شركة Bell وتم بناؤها بالأصل من أجل حواسيب تستخدم المعالج DEC – PDP-11 سنة 1972. تستخدم لغة C العديد من المفاهيم الهامة التي كانت موجودة في اللغتين BCPL و B بالإضافة إلى إضافة أنواع للمعطيات مع العديد من الإمكانيات الأخرى الهامة.

في البداية ، عرفت لغة C على أنها لغة التطوير الخاصة بنظام التشغيل Unix. أما حالياً فإننا نستطيع القول إن جميع نظم التشغيل الرئيسية هي نظم مكتوبة بلغة C أو لغة ++C . لقد أصبحت لغة C في العقدين الماضيين متاحة على معظم الحواسيب وتتصف بكونها مستقلة عن البنية الصلبة للحواسيب بالإضافة إلى كونها ذات تصميم دقيق ويمكن استخدامها لكتابة برامج قابلة للنقل بين معظم الحواسيب.

إن الاستخدام الواسع للغة C على العديد من الأنواع المختلفة من الحواسيب أدى إلى بعض الاختلافات بين نسخ لغة C التي يمكن أن تكون متشابهة ولكنها غالباً غير متوافقة فيما بينها. وهذا أدى إلى مشاكل مع مطوري البرامج الذين يحتاجون إلى كتابة برامج يمكن أن تعمل على عدة أنواع من الحواسيب. وظهرت نتيجة لذلك الحاجة إلى تعريف نسخة معيارية من لغة C.

تم إنشاء اللجنة التقنية X3JII في عام 1983 تحت إشراف اللجنة الوطنية الأمريكية للمعايرة الخاصة بالحواسيب ومعالجة المعلومات من أجل تحديد تعريف واضح ومستقل عن الحواسيب المستخدمة للغة البرمجة C. وتمت المصادقة على المعايرة التي أصدرت في عام 1989. بعدها قامت منظمة ANSI بالتعاون مع المنظمة العالمية للمعايرة ISO بتصميم المعايرة الخاصة بلغة C في مختلف أنحاء العالم وسميت هذه النسخة بنسخة ANSIC والتي هي قيد الاستخدام على مستوى العالم.

بسبب كون لغة C لغة معيارية واسعة الانتشار ومستقلة عن الآلة. فالتطبيقات المكتوبة بها هي تطبيقات يمكن تنفيذها مع بعض التعديلات البسيطة أو من دون أي تعديل على عدد كبير من الأنظمة الحاسوبية المختلفة.

تم بعد ذلك تطوير لغة ++C التي تُعد امتداداً للغة C من قبل Bjarne Stroustup في بداية الثمانينات، في مختبرات شركة Bell. تقدم لغة ++C العديد من الإمكانيات التي تسمو بلغة C وتعلو بها ولكن الأهم من ذلك كله هو بقدرات البرمجة غرضية التوجه O.O.P. لأن الأغراض (Objects) من المكونات البرمجية القابلة للاستخدام أكثر من مرة وهي قادرة على تمثيل العناصر من العالم الحقيقي المحيط بنا. ولأن استخدام أسلوب البرمجة الذي يعتمد على تجزئة المسألة المدروسة وعلى استخدام أساليب التصميم والبرمجة غرضية التوجه فإن هذا يجعل من مجموعات تطوير البرمجيات أكثر إنتاجية مما كانت عليه عند استخدام الأساليب البرمجية السابقة مثل أسلوب البرمجة المهيكلة. فالبرامج غرضية التوجه هي برامج سهلة الفهم وقابلة للتعديل والتصحيح بسهولة.

تُعد لغة ++C لغة هجينة (Hybrid language) فمعها يمكن كتابة برامج باستخدام أسلوب مشابه للبرمجة بلغة C أو باستخدام أسلوب غرضي التوجه أو كلاهما إذ تُعد البرمجة غرضية التوجه طريقة جديدة للوصول إلى الهدف من البرنامج أو هي طريقة جديدة في تكوين ملف المصدر وتسهيل السيطرة على البرامج الكبيرة.

تتميز لغة ++C بقابليتها على معالجة التطبيقات الكبيرة والمعقدة ومن الوسائل الجديدة في لغة ++C مفهوم ومبدأ الأصناف (Classes) التي تساعد في تقليل عدد تعليمات برنامج ++C مقارنة مع تعليمات C فضلاً عن إغناء اللغة بطابع الندرة والجمال.

ومن ميزات لغة ++C ما يلي:

1- لغة ++C لغة صغيرة : إذ تستخدم عدداً محدداً من الكلمات المحجوزة (Key words) وهذا أمر مرغوب في عمليات البرمجة، لأن البرامج التي تكتب بها سهلة ومختصرة.

2- لغة ++C لغة بنيوية (Structured language) : حيث يبين البرنامج مجموعة من تعليمات التحكم التي تهدف إلى إعطاء البرنامج صورة متماسكة وسهلة التداول.

3- لغة ++C غنية بالتوابع (Functions) : لغة ++C غنية بالتوابع الرياضية والحسابية وتوابع المعالجة الرمزية للنصوص وتوابع الإدخال والإخراج. الأمر الذي يسهل عملية بناء البرامج بها. وهذه التوابع جاهزة للاستخدام من دون الحاجة لإعادة كتابتها ثانية مما يوفر الوقت والجهد على المبرمجين.

4- لغة ++C لغة شاملة : تقع لغة ++C بين اللغات عالية المستوى مثل باسكال وفورتران وبيسك وكوبول وغيرها، واللغات منخفضة المستوى مثل لغة التجميع (Assembly) .

5- لغة ++C هي لغة نظام التشغيل Unix : يكتب نظام التشغيل Unix بلغة ++C أو C وهذا مصدر قوة للغة ويعطيها ميزة سرعة التنفيذ معه.

6- لغة ++C مستقلة عن الأجهزة إلى حد كبير : ويعني ذلك إمكانية نقل البرامج المكتوبة لجهاز Apple مثلاً إلى جهاز IBM بسهولة.

7- لغة ++C ذات نظام توزيع حركي اقتصادي للذاكرة: تمتلك ++C عدداً من التوابع الخاصة بالتوزيع الانتقائي الديناميكي للذاكرة (Memory dynamic allocation) ،

وبذا يتم التعامل مع جميع الأمكنة الشاغرة في الذاكرة بشكل أعظمي (Optimum) على الدوام.

8- لغة ++C لغة مرنة Flexible : تمتلك لغة ++C تعليمات قوية عديدة يعطي المبرمج حرية الاختيار والحركة عند بناء برامجه.

9- لغة ++C هي لغة OOP : حيث تعتمد لغة ++C على :

1) مبدأ الأصناف (Classes) وهو مبدأ هام في برمجة الكيانات.

2) مبدأ إعادة استخدام البرامج حيث تستفيد من البرامج المكتوبة مسبقاً وكما يمكن للبرنامج وراثة برامج أخرى مما يجعل البرنامج المكتوب صغيراً.

3) مبدأ الاستعمال المتعدد للأغراض فبالإضافة لكون لغة ++C هي لغة برمجية الكيان فإنها توفر وسيلة الاستعمال متعدد الأغراض للتوابع والأدوات.

3-2- أساسيات اللغة البرمجية:

ذكرنا في الفصل الأول الفقرة (1-4) بعض لغات البرمجة المنتمية إلى مستويات ثلاثة وهي لغات المستوى العالي مثل لغة ++C. واللغة التجميعية وهي ذات مستوى وسط بين لغات المستوى العالي والمستوى المنخفض (لغة الآلة) ولكن ما هي اللغة البرمجية؟

يمكن تعريف اللغة البرمجية الموضوعة من قبل الإنسان والمستخدمة لترميز المعلومات والمعطيات والتي يفهمها الحاسوب بعد أن تتم ترجمتها بوساطة المترجمات Compilers أو المفسرات Interpreters إلى لغة الآلة Machine language وتتكون كل لغة برمجية من العناصر التالية:

3-2-1- الرموز الأساسية (C++ symbols) :

يمكن تعريف الرمز Character بأنه أي حرف من حروف الهجاء (A→Z) أو رقم من الأرقام (0←9) أو علامة من العلامات مثل (+ , - , x , / , % , #) .

معظم أجهزة الحاسوب تستخدم كما نوهنا بذلك سابقاً نظام الشيفرة الأمريكية القياسية لتبادل المعلومات (ASCII code) . انظر الملحق A لمجموعة الشفرة الكاملة لهذا النظام.

تستخدم لغة C++ بعض أو كل مما يلي :

1- الحروف اللاتينية الكبيرة والصغيرة Latin literals

a,b,c,…,x,y,z and A,B,C,…,X,Y,Z

2- الأرقام العربية Arabic digitals وهي 9 , … , 2 , 1 , 0

3- الرموز الخاصة Special characters

+ - * / ; . $ 1) < > = ? " \ { } [ ] & % #

مثل علامات الترقيم الموجودة على لوحة مفاتيح الحاسوب وكذلك بعض الحروف التي لاتطبع. مثال رمز الانتقال إلى سطر جديد (\n) هو واحد من الرموز التي لاتطبع وهو مكون من الخط المائل (Back slash\) وحرف n ، وكذلك رمز ترك حقل خالٍ أفقي tab (\t) ورمز الإنذار (alarm) الذي ينتج صفيراً عند الطباعة (\a) ولطباعة الخط المائل \ نستخدم الرمز \\ .

3-2-2- الكلمات المحجوزة (Key words) :

وهي الكلمات القياسية المعرفة مسبقاً لمترجم C++ . وتستخدم للوظيفة المحددة لها فقط. أي لايمكن إعادة تعريفها أو استعمالها لغير ما خصصت له. انظر الملحق B الذي يبين الكلمات المفتاحية للغة C++ والتي عددها 52 كلمة محجوزة فقط. لذا تعد لغة C++ لغة صغيرة مقارنة مع اللغات الأخرى، فلغة أدا (Ada) على سبيل المثال تحتوي على 62 كلمة محجوزة.

فيما يلي بعض الكلمات المحجوزة:

for, while, do , continue, break.

3-2-3- الأسماء التعريفية (Identifiers):

يسمي المبرمج المواقع التخزينية في الذاكرة لقيم إما ثابتة أو متغيرة أو لتوابع حسب القواعد التالية:

1- يتكون الاسم من أحرف (Letter) وأرقام (Digital) وخطٍ تحتي (Under score) .

2- لايبدأ الاسم برقم ، ولايحوي فراغاً.

3- يجب أن لايكون أحد الأسماء المحجوزة (Key words) المتعلقة بلغة ++C .

4- تسمح اللغة بأي طول للاسم ولكن النظام المستخدم قد يفرض طولاً محدداً للأسماء.

5- لغة ++C حساسة لحالة الحرف: كبير أو صغير. وذلك بسبب اختلاف معاملة المترجم للحروف الكبيرة والصغيرة ولذلك فإن الاسمين : a1 و A1 مختلفان ، وكذلك Age و AGE و age تُعد أسماء مختلفة.

من المفيد استخدام أسماء ذات دلالة على الكينونة التي تسمى وبذلك نسهل قراءة البرنامج ونتجنب استخدام تعليقات توضيحية كثيرة.

فيما يلي أسماء صحيحة بلغة ++C :

My_name, a , x , salary, _name, employee

وأسماء غير صحيحة بلغة ++C :

7sum لوجود رقم في بداية الاسم.

Total.sum لوجود نقطة بين رموز الاسم.

NO # 1 لوجود الرمز # ضمن رموز الاسم.

my age لوجود فراغ ضمن رموز الاسم.

4-2-3- المتغيرات (Variables):

يمثل المتغير موقعاً للتخزين في الذاكرة يسميه المبرمج حسب شروط إعطاء الاسم في لغة ++C. والبيانات التي تخزن في هذا الموقع هي قيمة المتغير (value).

ويعطى المتغير قيمه ما إما عن طريق تعليمة التعيين (assignment statement) أو تعليمة الإدخال <<cin وسنتطرق لهما لاحقاً.

يجب التصريح أو الإعلان عن المتغيرات التي ستستخدم في البرنامج قبل استخدامها وذلك لإعطاء الحاسوب (المترجم) فكرة عن أسماء وأنواع البيانات التي تحملها تلك المتغيرات وبالتالي حجمها.

المتغيرات في لغة ++C على أنواع وتختلف فيما بينها بالحيز الذي تحتاجه من الذاكرة لتخزينها.

الجدول (3-1)التالي يبين أنواع المتغيرات المقبولة في لغة ++C وحجم كل منها بالبايت والقيمة التي يخزنها كل نوع، انظر الملحق (D) الذي يبين الأنواع المختلفة والأساسية للبيانات المعرفة في لغة ++C . وكذلك الأنواع المشتقة من قبل المبرمج باستخدام الأنواع الأساسية.

جدول (3-1) أنواع متغيرات لغة ++C

Data type	size	Value
Unsigned short int عدد صحيح قصير غير مؤشر	2 byte	0 to 65,535
Short int صحيح قصير مؤشر	2 byte	-32,768 to 32,767
صحيح Int	2 byte	-32,678 to 32,767
Unsigned int عدد صحيح غير مؤشر	2 byte	0 to 65,535
صحيح طويل long int	4 byte	-2,147,483,648 to 2,147,483,647
Unsigned long int صحيح طويل غير مؤشر	4 byte	0 to 4,294,967,295
حقيقي Floot	4 byte	1.2e-38 to 3.4e38
حقيقي مضاعف الدقة Double	8 byte	2.2e – 308 to 1.8e308
رمزي Char	1 byte = 8 bit	256 character value

في معظم الحواسيب الشخصية يكون الحيز الذي يحتاجه النوع short int مساوياً للحيز الذي يحتاجه نوع int أما النوع long int فيحتاج إلى ضعف الحجم الذي يحتاجه النوع int.

ملاحظة (1) : تحديد نوع المتغير هام جداً في لغة ++C بحيث يتناسب النوع المعلن للمتغير مع قيمة البيانات المطلوب تخزينها فيه.

فمثلاً لو صرح عن متغير A على أنه unsigned int وبالنظر للجدول السابق نلاحظ أن القيمة العظمى للمتغير A هي 65535 ولو أضيفت القيمة 1 واحد إلى المتغير A فإن القيمة الجديدة التي ستخزن في A هي 0 صفر بدلاً من 65535+1. تماماً كما يعمل عداد السيارة إذ عندما يصل العداد إلى قيمته العظمى يعود ويأخذ القيمة صفر بزيادة العداد بواحد.

ملاحظة (2) : هناك متغيرات من نوع void وهي لاتملك أية قيمة ولا يحجز لها أي حجم تخزيني في الذاكرة.

3-2-5- الثوابت (Constant) :

الثوابت هي مواقع تخزين في الذاكرة تسمى من قبل المبرمج وتستخدم لتخزين قيم ثابتة لايمكن تغييرها في أثناء تنفيذ البرنامج، على عكس المتغير الذي يمكن تغيير قيمته.

الشكل العام للتصريح عن الثابت هو :

```
const  data_type  constant_name = value;
```

حيث:

const : كلمة محجوزة في لغة ++C تخبر الحاسوب بأن لايقبل تغيير القيمة المخزنة في موقع الذاكرة المصرح عنه بـ const.

constant_name : هو اسم للثابت يعطى من قبل المبرمج.

data_type : نوع البيانات المخزنة في الثابت.

value : القيمة المخزنة في الثابت.

مثلاً الثابت π يمكن التصريح عنه بإعطائه اسماً مثل PI على الشكل التالي:

const float PI = 3.14;

التعليمة السابقة تصرح عن ثابت اسمه PI قيمته 3.14 ونوعه حقيقي، أما التعليمة التالية : ;PI = 2.14 فهي تعليمة خاطئة لأنها تحاول تعيين قيمة جديدة للثابت PI .

6-2-3- المعطيات غير العددية (الرمزية) (non-numeric or string constant):

المعطيات الرمزية هي سلسلة من الرموز المقبولة في اللغة من أحرف وأرقام ورموز خاصة توضع بين علامتي اقتباس أو ما يسمى بالحاصرات العلوية المزدوجة " " (double quotation marks) .

أما طول المعطيات الرمزية فهو عدد الرموز المحصورة بين الحاصرات المزدوجة. مثلاً : طول القيمة الرمزية "learn C++" هو 9 رموز حيث يحسب الفراغ رمزاً واحداً. كل المعطيات الرمزية وإن استخدمت أرقاماً حسابية داخلها إلاّ أنها لاتحمل أية قيمة حسابية وليس لها معنى حسابي.

تستعمل لغة C++ الحاصرات العلوية المفردة ' ' بصورة مختلفة عن الحاصرات المزدوجة " " ، حيث تستخدم الحاصرات المفردة لوضع رمز واحد فقط داخلها مثل 'a' والذي يختلف عن "a".

يتم تخزين أي من المعطيات الرمزية في بايت واحد من الذاكرة، وبما أن البايت مكون من 8 بت فإنه بمقدور البايت تخزين 2^7 أو 256 رمزاً مختلفاً من الحروف أو الأرقام أو الرموز الخاصة. وتستخدم في الحواسيب جداول بشيفرات رقمية لهذه الحروف

والأرقام والرموز الخاصة، أشهرها EBCD1C, ASC11 وهما موجودان كملاحق في الكتاب، الملحق C,A.

3-3- التعبير (Expression):

يمكن تعريف التعبير بأنه مجموعة من الحدود يربط بينها إما :

a- عوامل رياضية (Arithmetic operator) ويسمى عندها تعبير رياضي :

مثال : $2 * y - b + c$

هو تعبير رياضي يحوي ثلاثة حدود (الحد يمكن أن يكون عدد ثابت أو متغير) هي : $2 * y$ و b و c .

وثلاثة عوامل رياضية هي * للدلالة على عملية الضرب.

و – للدلالة على عملية الطرح.

و + للدلالة على عملية الجمع.

حيث العامل هو رمز يأمر المترجم بأداء عمل ما.

أمثلة لتعابير رياضية وما يقابلها بلغة C++ .

1- xy أو x.y أو x×y يقابلها بلغة C++ x * y

2- x/y أو $\frac{x}{y}$ أو x÷y يقابلها بلغة C++ x/y قسمة صحيحة

3- $\frac{a+b}{a-b}+c$ يقابلها بلغة C++ (a+b)/(a-b)+c

4- $2r - 6.5$ يقابلها بلغة C++ 2.0 * r – 6.5

5- $\sin(3.14 \times 15)$ يقابلها بلغة C++ sin(3.14*15)

b- أو عوامل مقارنة حسابية (Arithmetic comparison) : مثل تعبير المقارنة الحسابية التالي: $x < y$

c- أو عوامل المقارنة المنطقية (Boolean comparison) : مثل تعبير المقارنة المنطقية التالي: x && y

الجدول (3-2) التالي يبين جميع العوامل الرياضية المستخدمة بشكلها المقبول في لغة C++.

جدول (3-2) **العوامل الرياضية في** C++

تعبير بلغة C++	التعبير الرياضي	العامل الرياضي	نوع العملية في C++
a + 7	a + 7	+	الجمع
P – 2	P – 2	-	الطرح
z * y	zy or z×y or z.y	*	الضرب
x/y	x/y or $\frac{x}{y}$ or x÷y	/	القسمة الصحيحة
r%s	r mod s	%	باقي القسمة

ملاحظة (3) : قسمة الأعداد الصحيحة تأخذ الناتج الصحيح فقط ويهمل الباقي. أي 21/5 يعطي 4 . بينما يمكن الحصول على الباقي باستخدام العامل % حيث 21%5 هو 1.

ولكن لاستخدام العامل % (باقي القسمة) في لغة c++ يجب أن يكون كلا الحدين اللذين على يسار ويمين العامل من النوع الصحيح، بغض النظر عن نوع المتغير الموجود في الطرف الأيسر من تعليمة التعيين أي لو كان لدينا التعليمات التالية:

```
float a=15;
float b=7;
int z;
```

عندئذ فإن:

z = a+b → z = 22

z = a-b → z = 8

z = a/b → z = 2

z = a%b → z = لاتنفذ

الجدول (3-3)التالي يبين عوامل المقارنة الحسابية.

جدول (3-3) عوامل المقارنة الحسابية في C++

C++ Comparison operator		arithmetic Comparison expression	C++ arithmetic Comparison expression
Greater than	>	A > B	A > B
Less than	<	A < B	A < B
Greater or equal	> =	A ≥ B	A> = B
Less or equal	< =	A ≤ B	A< = B
not equal	! =	A ≠ B	A! = B
equal	= =	A = B	A = = B

الجدول (3-4)التالي يبين عوامل المقارنة المنطقية:

جدول (3-4) عوامل المقارنة المنطقية

boolen comparison operator		C++ boolen comparison operator	boolen comparison expression	C++ boolen comparison expression
نفي	Not	!	A not B	A ! B
أو	or	||	A or B	A || B
و	and	&&	A and B	A && B

ملاحظة (4) : قيمة التعبير المنطقي تساوي الواحد إذا كانت الإجابة صح (True) وتساوي الصفر إذا كانت الإجابة خطأ (False).

الجدول (3-5) التالي يبين جدول الحقيقة لكل من عوامل المقارنة المنطقية.

الجدول (3-5) جدول الحقيقة لعوامل المقارنة المنطقية

a	b	a||b
0	0	0
0	1	1
1	0	1
1	1	1

جدول الحقيقة للمعامل or

a	!a
0	1
1	0

جدول الحقيقة للمعامل not

a	b	a&&b
0	0	0
0	1	0
1	0	0
1	1	1

جدول الحقيقة للمعامل and

انظر الملحق (C) الذي يبين جميع العمليات في لغة ++C مرتبة حسب أسبقية التنفيذ.

3-4- تعليمة التعيين (Assignment) :

وهي على الشكل التالي :

variable = expression;

حيث تقدر قيمة التعبير (expression) أولاً ثم تعيين القيمة الناتجة للمتغير (variable) أي أن الصيغة في الطرف الأيمن تحسب وتعين للمتغير الوحيد في الطرف الأيسر علامة المساواة " = " تسمى معامل التعيين (assignment operator) .

مثال : تعليمة التعيين ;y = 2 + x

تعني اجمع لقيمة المتغير x القيمة 2 ثم عين الناتج للمتغير y ، أما تعليمة التعيين ;2 = x + y فهي تعليمة خاطئة لأن الطرف الأيسر هو عدد وليس متغير.

التعليمات التالية هي تعليمات تعيين صحيحة:

```
Area = 2 * r * 3.14;
Sum = 0;
Average = sum/n;
y = z;
y = X > K;
```

والتعليمات التالية هي تعليمات تعيين غير صحيحة:

;A – 1 = 10 الطرف الأيسر تعبير حسابي وليس متغيراً وحيداً.

F = b * 5 تعليمة التعيين لاتنتهي بالفاصلة المنقوطة.

3-4-1- تعليمة التعيين المتعدد:

إذا كان لدينا عدة متغيرات وأردنا تعيين القيمة نفسها لها أي :

K = 5;   L = 5;   m = 5;

عندئذ يمكننا أن نكتب اختصاراً:

$$K = L = m = 5;$$

تدعى هذه الصيغة بالتعيين المتعدد (Multiple assignment statement) حيث يقوم الحاسوب بتنفيذ مؤثر التعيين من اليمين إلى اليسار أي يتم تعيين القيمة 5 للمتغير m أولاً ثم تعيين قيمة المتغير m وهي 5 إلى المتغير L ثانياً . ثم يتم أخيراً تعيين قيمة المتغير L للمتغير K.

مثال (1) :

بفرض أنه لدينا n = 10 و m = 3 و K عندئذ فإن K تأخذ القيمة الموافقة لكل تعليمة تعيين مقابلة.

$$\begin{array}{lll} K = n == m; & \Rightarrow & K = 0 \\ K = !(n == m); & \Rightarrow & K = 1 \\ K = n > m; & \Rightarrow & K = 1 \\ K = !(n > m); & \Rightarrow & K = 0 \end{array}$$

فيما يلي تعليمات مقبولة في لغة C++ :

```
int myage = 5;
int  temp;
temp = myage + 2;
myage = temp;
```

يمكن كتابة التعليمتين الثالثة والرابعة بتعليمة واحدة هي:

```
myage = myage + 2;
```

أو بشكل مختصر أكثر كما يلي :

```
myage + = 2;
```

هذه التعليمة تعني أضف القيمة 2 الموجودة إلى اليمين إلى محتوى المتغير الموجود إلى اليسار ثم عين القيمة الناتجة للمتغير الموجود في اليسار ويلفظ كما يلي myage زائد يساوي.

بفرض أن القيمة الآنية لـ myage هي 5 فإن ناتج التعليمة السابقة هو تعيين القيمة 7 للمتغير myage.

3-5- عوامل التعيين المركبة (Compound assignment operators)

تسمح لغة ++C باستخدام عوامل التعيين المركبة التالية :

زائد يساوي	+ =
ناقص يساوي	- =
ضرب يساوي	* =
قسمة صحيحة يساوي	/ =
باقي القسمة يساوي	% =

مثال :

إذا كانت قيمة X الأولية هي 10 فإن :

زيادة قيمة X بـ 2	$X+=2;\ \Rightarrow\ X=12$
إنقاص قيمة X بـ 2	$X-=2;\ \Rightarrow\ X=8$
ضرب قيمة X بـ 2	$X*=2;\ \Rightarrow\ X=20$
قسمة قيمة X على 2	$X/=2;\ \Rightarrow\ X=5$

3-6- عوامل التزايد والتناقص (Increment & decrement operators):

بفرض التعليمتين التاليتين:

```
X = X + 1;
X = X – 1 ;
```

فإن التعليمة الأولى تعني زيادة قيمة المتغير X بمقدار واحد أما التعليمة الثانية فتعني إنقاص قيمة المتغير X بمقدار واحد.

يمكن كتابة التعليمتين السابقتين باستخدام عوامل التعيين المركبة كما يلي:

```
X + = 1;
X - = 1;
```

تسمح لغة C++ بطريقة أخرى مختصرة وأسهل في التعامل مع المتغيرات التي تتزايد أو تتناقص بمقدار واحد باستخدام عامل التزايد ++ أو عامل التناقص - -

العامل	مثال	مثال مكافئ في C++
++	X = X + 1;	+ +X; X + +;
– –	X = X – 1;	– –X; X – –;

7-3- استخدام عوامل التزايد والتناقص ضمن التعبير:

في الفقرة السابقة ذكرنا أنه لزيادة قيمة متغير بمقدار واحد يمكننا كتابة التعليمات الصحيحة التالية:

```
X = x + 1;
X + = 1;
X++;
++X;
```

التعليمتان الثالثة والرابعة لهما التأثير نفسه وهو زيادة قيمة المتغير X بمقدار واحد رغم استخدام عامل التزايد مرة قبل اسم المتغير (preincrement) كما في التعليمة الرابعة ومرة بعد اسم المتغير (postincrement) كما في التعليمة الثالثة.

ولكن لو استخدم اسم المتغير ذي عامل التزايد ++ أو التناقص – – ضمن تعبير ما فإن موقع عامل التزايد أو التناقص قبل المتغير أو بعد المتغير يؤثر في النتيجة كما هو موضح في الأمثلة التالية :

```
Sum + = ++ count;
Sum + = count++;
```

التعليمة الأولى تزيد القيمة الآنية للمتغير count بواحد ثم تجمع الناتج مع قيمة المتغير sum وتعين الناتج كقيمة جديدة للمتغير sum.

أما التعليمة الثانية فتجمع القيمة الآنية للمتغير count قبل زيادته بواحد إلى قيمة المتغير sum وتعين الناتج كقيمة جديدة للمتغير sum ثم تزيد قيمة المتغير count بواحد.

لو كان لدينا count = 20 و sum = 10

فإن التعليمة الأولى تزيد أولاً قيمة count بواحد فتصبح count = 21 ثم تجمعها مع قيمة sum وتعين الناتج لـ sum أو :

Sum = 10 + 21 = 31

بينما في التعليمة الثانية تجمع أولاً قيمة count وهي 20 مع sum وهي 10 وتعين الناتج كقيمة جديدة للمتغير sum تساوي

Sum = 10 + 20 = 30

ثم تضيف قيمة المتغير count بواحد فيصبح أيضاً 21 .

مثال آخر : لو كان لدينا التعليمتان التاليتان:

```
i = 7;
j = 4 + - - i;
```

نتيجة تنفيذهما هي : j = 10, i = 6

أي أن استخدام عامل التزايد أو التناقص قبل المتغير (من جهة اليسار) يجعل الحاسوب يقوم بتنفيذ عامل التزايد أو التناقص أولاً ثم بقية عوامل التعبير وذلك لأن التعليمتين السابقتين هما اختصار لما يلي :

```
i=7;
i = i-1;
j = 4+i;
```

بفرض لدينا التعليمتان التاليتان:

```
i=7;
j = 4+i--;
```

فإن تنفيذ ذلك يعطي النتيجة التالية :

j = 11 , i = 6

أي أنه إذا استخدم عامل التزايد أو التناقص بعد المتغير (من جهة اليمين) عندئذ يقوم الحاسوب بتنفيذ عوامل التعبير أولاً ثم عامل التزايد أو التناقص وذلك لأن التعليمتين السابقتين هما اختصار لما يلي:

```
i = 7;
j = 4 + i;
i = i – 1;
```

مثال:

```
int x,y,z;
x = y = z = 0;
x = ++ y + ++z;
```

⇐ في هذه الحالة تكون

x = 2 , y = 1 , z = 1

```
x = y++ + z++
```

أولوية جمع واحد إلى z تأتي متأخرة إذ يتم جمعها أولاً إلى x

جمع واحد إلى y مؤجل لبعد إعطاء قيمتها الأولى إلى x

⇐ في هذه الحالة تكون

x = 2 , y = 2 , z = 2

لأن x هي حاصل جمع قيمتي y و z قبل أن تزداد كل منهما بمقدار واحد

```
x = ++y + z++;
```

⇐ الآن

x = 5 , y = 3 , z = 3

لأن x هي حاصل جمع y بعد زيادتها بواحد مع آخر قيمة لـ z قبل زيادتها بواحد

```
x = y-- + --z;
```

⇐ في هذه الحالة :

x = 5 , y = 2 , z = 2

لأن x هي حاصل جمع y قبل إنقاصها بواحد مع قيمة z بعد إنقاصها بواحد

8-3- أولوية تنفيذ عوامل لغة ++C في تعابير حسابية ومنطقية:

يُقصد بأولوية التنفيذ الأسس المستخدمة في معالجة التعابير الحسابية والمنطقية حسب منطق الحاسوب واللغة البرمجية المستخدمة.

الجدول (3-6) التالي يوضح أولوية تنفيذ بعض العوامل في لغة البرمجة ++C.

جدول (3-6) أولوية تنفيذ بعض العوامل في لغة ++C

Operator **العامل**	**Order** **الترتيب**	**Associative** **طريقة التنفيذ في حال تكرار العامل في التعبير**
الأقواس ( )	أولاً	من اليسار إلى اليمين والأقواس الداخلية أولاً
! ++,-- (befor)	ثانياً	من اليسار إلى اليمين
* ,/, %	ثالثاً	من اليسار إلى اليمين
+,-	رابعاً	من اليسار إلى اليمين
<, < = , > , > =	خامساً	من اليسار إلى اليمين
= = , ! =	سادساً	من اليسار إلى اليمين
&&	سابعاً	من اليسار إلى اليمين
||	ثامناً	من اليسار إلى اليمين
+ = - = * = / =	تاسعاً	ينفذ الطرف الأيمن ويعين الناتج للطرف الأيسر
++, -- (after)	أخيراً	من اليمين إلى اليسار

انظر الملحق C الذي يبين جميع العمليات في لغة ++C مرتبة حسب أسبقية التنفيذ.

من مطالعة الملحق (C) نلاحظ أن لغة ++c لغة غنية بالعديد من المعاملات.

نلاحظ أيضاً أن معامل الضرب * والقسمة / وباقي القسمة % لها أسبقية على معاملات الجمع + والطرح – أي أنها تنفذ أولاً . مثلاً :

42-5 * 2

تنفذ كالتالي :

42-(5 * 2) = 42-10 = 32

وكل المعاملات الحسابية لها أسبقية في التنفيذ على معاملات التعيين والخرج، على سبيل المثال التعليمة التالية:

n = 42-3 * 2;

سوف تعين القيمة الناتجة من حساب التعبير 2*3-42 للمتغير n. لذا يستدعى أولاً ا لمعامل * لحساب 2 * 3 ثانياً يستدعى المعامل – لحساب 6-42 ثم يستدعى المعامل = ليعين القيمة 36 للمتغير n.

نلاحظ أيضاً من الملحق C أن معامل النفي الأحادي له مستوى أسبقية 15 ومعامل الضرب الثنائي * له مستوى أسبقية 13 ، لذا عملية النفي تنفذ قبل عملية الضرب ولذلك فالتعبير الجبري a *-b ينفذ كالتالي a* (-b) .

العمود الثاني في الملحق (C) والمسمى ASSOC اختصاراً لكلمة الحاق (Association) يخبرنا ماذا يحدث إذا وجد أكثر من معامل بالأسبقية نفسها في تعبير واحد، على سبيل المثال + و - لهما مستوى الأسبقية 12 نفسه وإلحاق من اليسار لذلك فإن المعاملات تنفذ من اليسار إلى اليمين أيهما يأتي أولاً . ففي التعبير التالي :

7-4 + 5

نطرح أولاً 4 من 7 ثم نجمع 5 للناتج :

(7-4) + 5 = 3 + 5 = 8

أما العمود المسمى الحالة Arity فيعلمنا إذا كان المعامل أحادياً أو ثنائياً. أحادي unary يعني أن المعامل ينفذ على طرف واحد مثل الزيادة ++ هي أحادي ++n يعمل

على متغير واحد فقط هو n . أما الثنائي binary فيعني أن المعامل ينفذ على طرفين ، مثلاً الجمع + هو ثنائي حيث a+b ينفذ على متغيرين هما a و b.

مثال (1) :

بفرض أن لدينا :

```
int  n = 20;
int  m = 7;
int  k;
```

عندئذ فإن :

```
k = (n = = m) && (n > m);    ⇒  k = 0 && 1    ⇒  k = 0
k = (n = =  m) && (n < m);   ⇒  k = 0 && 0    ⇒  k = 0
k = (n! =  m) && (n < m);    ⇒  k  = 1 && 0   ⇒  k = 0
k = (n! = m) && (n > m);     ⇒  k = 1 && 1    ⇒  k = 1
k = (n! =  m) ||  (n > m);   ⇒  k = 1 || 1    ⇒  k = 1
k = (n! =  m) || (n < m);    ⇒  k = 1 || 0    ⇒  k = 1
k = (n = = m) || (n < m);    ⇒  k = 0 || 0    ⇒  k = 0
k = (n = = m) || (n > m);    ⇒  k = 0 || 1    ⇒  k = 1
```

مثال (2):

بفرض لدينا :

```
int i = 14, j = 5, k = 3;
int z1, z2, z3;
```

بين أولويات تنفيذ العمليات في التعابير التالية :

```
Z1 = (i+j)*2 – i/3;
Z2 = i+j*2 – i/3;
Z3 = i%j + k*5;
```

التعبير Z1 :

الخطوة الأولى : يتم حساب ما داخل أقواس أي جمع قيمة i مع قيمة j وهذا يعطي 19.

الخطوة الثانية : يتم ضرب القيمة 2 بناتج الخطوة الأولى وهذا يعطي 38 = 19 × 2

الخطوة الثالثة : يتم قسمة قيمة i على 3 قسمة صحيحة وهذا يعطي 4.

الخطوة الرابعة : يتم طرح ناتج الخطوة الثالثة من ناتج الخطوة الثانية وهذا يعطي 34

الخطوة الخامسة : تعيين ناتج الخطوة الرابعة للمتغير z1

z1=19*2 – i/2 = 38 – i/2 = 38 – 4 = 34

التعبير Z2 :

الخطوة الأولى : يتم حساب حاصل ضرب 2 بقيمة j وهي 5 وهذا يعطي 10.

الخطوة الثانية : يتم قسمة i قسمة صحيحة على 3 وهذا يعطي 4

الخطوة الثالثة : يتم جمع ناتج الخطوة الأولى وهو 10 مع قيمة i وهذا يعطي 24.

الخطوة الرابعة : يتم طرح ناتج الخطوة الثانية من ناتج الخطوة الثالثة وهذا يعطي 20

الخطوة الخامسة : تعيين ناتج الخطوة الرابعة للمتغير z2

Z2 = i + 10 – i/3 = i + 10 – 4 = 24 – 4= 20

التعبير Z3

الخطوة الأولى : يتم حساب باقي قسمة i على j وهذا يعطي 4.

الخطوة الثانية : يتم حساب حاصل ضرب 5 بقيمة k وهذا يعطي 15.

الخطوة الثالثة : يتم حساب حاصل جمع ناتج الخطوة الثانية مع ناتج الخطوة الأولى وهذا يعطي 19

الخطوة الرابعة : تعيين ناتج الخطوة الثالثة للمتغير z3

Z3 = 4 + k * 5 = 4 + 15 = 19

3-9- برنامج بسيط في لغة C++ :

المثال التالي يوضح الأجزاء الرئيسية لبرنامج بسيط في لغة ++C ولنسمي الملف الذي نخزن فيه هذا البرنامج بالاسم Hello.cpp حيث cpp هو امتداد الملف المكتوب بلغة ++C.

```
# include < iostream.h >
// this program prints " Hello, student."
main ()
  {
      cout << " Hello, student. \n";
      return 0;
  }
```

يبدأ الملف Hello,cpp بالسطر الأول الذي يبدأ بالرمز # وهو إشارة إلى المعالج المسبق (Preprocessor) بتضمين (Include) الملف الرأسي <iostream.h > إلى البرنامج السابق في سطره الأول. الملف ذو الامتداد h هو ملف رأسي (header) .

ماهو المعالج المسبق: المعالج المسبق هو برنامج آخر يستدعى عند تشغيل المترجم (Compiler) ولن نحتاج لاستدعائه مباشرة لأنه يعمل بصورة تلقائية كلما شغلنا المترجم.

وظيفة هذا البرنامج هي قراءة الكود المصدري (source code) للبرنامج بحثاً عن أسطر تبدأ بالرموز # . ومن ثم يأتي دور Include وهي كلمة مفتاحية تقول للمعالج المسبق إن ما يأتي بعدها هو اسم الملف المفترض وضعه مكان هذه التعليمة. الأقواس الزاوية > و < ليست جزءاً من اسم الملف ولكنها تستخدم لبيان أن هذا الملف من مكتبة لغة ++C القياسية.

يتضح من اسم الملف Input /output stream.header أن وظيفته تشغيل أوامر الإدخال والإخراج الموجودة داخل البرنامج المصدر.

إذن السطر الأول يقوم بإدراج الملف iostream.h داخل ملف البرنامج Hello.cpp بحيث تبدو مكتوبة معاً. لذلك يتعامل المترجم مع الملف Hello.cpp الذي يحوي الملفين معاً من دون ملاحظة ما حصل.

السطر الثاني هو عبارة عن ملاحظة أو تعليق Comment . يبدأ سطر التعليق إما بالشرطتين المائلتين // وعندها يجب أن ينتهي التعليق في سطر واحد فقط. أو بالرمز */ وينتهي بالرمز /* ويمكن عندها أن يمتد التعليق ليشكل أكثر من سطر واحد.

توضع الملاحظات أو التعليقات في البرنامج لتزويد القراء بالشرح والتوضيح عن خطوات البرنامج وليصبح بذلك أكثر وضوحاً مع العلم أن الملاحظات لايهتم لها المترجم ولاتؤثر في تنفيذ البرنامج. وينصح دائماً باستخدام التعليق الملائم والمفهوم للبرنامج بحيث يوضح عمل كل مقطع من الكود من دون إسراف في ذلك.

يحوي السطر الثالث اسم التابع الرئيسي () main. وهذا التابع يجب أن يكون موجوداً في كل برنامج ++C . وهو يخبر المترجم من أين يبدأ تنفيذ البرنامج ، حتى لو لم يكن هذا التابع هو التابع الأول في ملف البرنامج.

القوسان () التابعان لاسم التابع main يجب أن يكونا موجودين أيضاً ليدلا على أن main عبارة عن كتلة برمجية تسمى بتابع function.

السطر الرابع والسابع يحويان على القوسين } و { الأول استهلالي والآخر ختامي يوجد بين هذين القوسين جسم التابع () main وهما لازمان لكل برنامج بلغة ++C.

السطر الخامس يحوي التعليمة التالية :

```
Cout << " Hello. Student. \n";
```

وهي تعني إرسال العبارة Hello.student إلى الغرض cout وتنطق (see out). هذا الغرض هو مجرى الخرج القياسي (standard output stream) ، عادة يكون شاشة

الحاسوب. والعامل >> هو عامل الإخراج (insertion operand) . الرمز (n\) هو رمز الانتقال إلى سطر جديد. لاحظ أنه يتكون من الخط المائل '\' وحرف 'n' .

وجود هذا الرمز في نهاية التعليمة بين علامتي اقتباس (double quotation mark) يخبر الحاسوب بأن يبدأ سطراً جديداً بعد طباعة الحرف السابق للرمز n\ أي بعد الانتهاء من طباعة Hello. Student وهذا يعني نهاية السطر الحالي.

السطر السادس يحوي التعليمة ;return 0 . وهذه التعليمة تنهي تنفيذ البرنامج ويتم العودة إلى نظام التشغيل للحاسوب. الرقم 0 يستخدم كإشارة بأن البرنامج انتهى بنجاح.

ملاحظة (5) : الرمز >> يسمى عامل الخرج (output operator) أو معامل الإدماج (insertion operation) ، فهو يدمج أو يضع العبارة المراد إخراجها في مجرى الخرج (output stream).

ملاحظة (6) : الرمز n \ في نهاية العبارة يعني الانتقال إلى سطر جديد.

وجود هذا الرمز في أي مكان في الرسالة المراد طباعتها يسبب نهاية السطر الحالي في الخرج وبداية سطر جديد. لاحظ أن كلاً من الرمز >> و n\ يتكونان من حرفين متجاورين ليس بينهما فراغ أو مسافة.

ملاحظة (7) : لاحظ وجود الفاصلة المنقوطة ; (semicolon) في نهاية كل من السطر الخامس والسادس. كل تعليمة أو أمر في ++C يجب أن ينتهي بالفاصلة المنقوطة.

ملاحظة (8) : ليس بالضرورة أن تكون كل تعليمة في سطر مستقل، يمكننا وضع مجموعة من التعليمات في سطر واحد ويمكن أيضاً وضع تعليمة واحدة في أكثر من سطر ، المهم أن كل تعليمة يجب أن تنتهي بالفاصلة المنقوطة.

10-3 تمارين

1- اكتب العلاقة الرياضية التالية بشكلها المكافئ في لغة ++c

$$x = \frac{5a + 10b - (xb/4)}{(2 + b + ab) - (2b)}$$

بفرض أن a=4 , b=2 , x=5 متغيرات صحيحة. أوجد قيمة التعبير.

العلاقة المكافئة بلغة ++c هي:

```
x = (5 * a + 10 * b – (x * b/4)) / ((2 + b + a*b) – (2*b));
 (13)   (3)   (5)    (4)  (6)    (1)  (2)  (12)   (8)    (9)  (7)  (11)  (10)
  = (5 * 4 + 10 * 2 – ( 10/4 )) / (( 2 + 2 + 4 * 2) – (2 * 2));
  = (5 * 4 + 10 * 2 – 2) / ((2 + 2 + 8) –4);
  = (20 + 20 – 2) / (12-4);
  = 38 / 8;
  = 4
```

2- اكتب العلاقة الرياضية التالية بشكلها المكافئ في لغة ++c

$$x = a^2 - 5b + \frac{10d - 5d^2 + 1}{(12d + 4b)/(5b + 4)} - 6b$$

بفرض أن d = 5, b = 4 , a = 2 متغيرات صحيحة. أوجد قيمة التعبير.

العلاقة المكافئة بلغة ++c هي:

```
x = a*a – 5*b + ( 10*d – 5 * d*d + 1) / (( 12*d + 4*b ) / (5*b + 4)) – 6*b;
(19) (12) (16) (13) (17)      (7) (10)  (8)    (9)  (11)    (14)      (1)  (3)  (2)     (6)   (4)  (5)      (18)  (15)
```

وبتنفيذ الحسابات الجزئية بالترتيب المكتوب نحصل على قيمة x = - 64.

3- بفرض أن i = 2 , j = 3 , k = 4 متغيرات صحيحة . بيّن أولويات تنفيذ كل من التعليمات التالية ونتائجها.

```
z1 = i * ( 7 + (j+3) / 2) – k;
z2 = 17% (k-i) * j + 6;
z3 = i + j – 5 * (k-3) / j – 5;
```

الحل:

```
z1 = i * ( 7 + ( j + 3 ) / 2 ) – k;
  (6) (4)   (3)   (1)   (2)   (5)
```

بالتعويض:

```
   = 2 * ( 7 + ( 3 + 3 ) / 2 ) – 4;
   = 2 * ( 7 + 6/2 ) – 4;
   = 2 * ( 7 + 3 ) – 4;
   = 2 * 10 – 4;
   = 20 – 4;
   = 16;
z2 = 17 % ( k – i) * j + 6 ;
  (5)  (2)   (1)  (3) (4)
```

بالتعويض

```
   = 17 % 2 * 3 + 6 ;
   = 1 * 3 + 6 ;
   = 3 + 6 ;
   = 9 ;
z3 = i + j – 5 * ( k – 3 ) / j – 5 ;
  (7) (4) (5) (2)   (1)   (3) (6)
```

بالتعويض:

```
   = 2 + 3 – 5 * ( 4 – 3 ) / 3 – 5 ;
   = 2 + 3 – 5 * 1 / 3 – 5 ;
   = 2 + 3 – 1 – 5 ;
   = 5 – 1 – 5 ;
   = 4 – 5 ;
   = – 1 ;
```

4- اكتب تعليمة واحدة بلغة ++c تحقق ما يلي:

a- تطرح مجموع x و y من z وبعد ذلك تزيد قيمة y بمقدار واحد.

b- تنقص قيمة المتغير n بواحد ثم تضيفه للمتغير z.

c- تضرب قيمة المتغير n بالمتغير m ثم تضيف واحد للمتغير n.

d- تزيد قيمة المتغير n ثم تطرحه من المتغير m.

a) z - = (x + y++);
b) z + = - - n;
c) m * = n++ ;
d) m - = ++ n;

5- بفرض لدينا i = 3 و j = 8 و k_1,k_2,k_3 متغيرات صحيحة و c = 'w' متغير حرفي بين أولويات تنفيذ التعليمات التالية ونتائجها:

```
k1 = ( i > = 0 ) + ( i ! = j ) ;
k2 = ( i + 5 * j ) < = ( 30 / i ) ;
k3 = i + j + 3 <j + i ! = 30 ;
'v' = = ( c – 1 ) ;
```

الحل:

```
k1 = ( i > = 0 ) + ( i ! = j ) ;
  (4)    (1)     (3)    (2)
   = ( 3 > = 0 ) + ( 3 ! = 8 ) ;
   = 1 + 1 ;
   = 2 ;
k2 = ( i + 5 * j ) < = ( 30 / i ) ;
  (5)   (2) (1) (4)      (3)
   = ( 3 + 5 * 8 ) < = ( 30 / 3 );
   = ( 3 + 40 ) < = ( 30 / 3 );
   = 43 < = 10 ;
   = 0
k3  = i  +  j + 3 <j + i ! = 30 ;
   (6)  (1)  (2) (4)(3)  (5)
   = 11 + 3 < 8 + 3 ! = 30 ;
   = 14 < 11 ! = 30 ;
   = 0 ! = 30 ;
   = 1
```

```
'v' = = ( c – 1 ) ;
    (2)      (1)
    = = ( w – 1 ) ;
    = = v ;
```

قيمة التعبير الأخير محققة أي واحد لأن شيفرة v هي أقل من شيفرة w بواحد. وذلك في الشيفرة العالمية المعتمدة في الحاسوب ASC11.

6- لو كان لدينا العلاقة التالية :

$$y = a\,x^3 + 7$$

أشر إلى التعليمة المكافئة لها بلغة c++ إن وجدت فيما يلي:

```
a)     y = a * x * x * x + 7 ;
b)     y = a * x * x * ( x + 7 ) ;
c)     y = ( a * x ) * x * ( x + 7 );
d)     y = ( a * x ) * x * x + 7 ;
e)     y = a * ( x * x * x ) + 7 ;
f)     y = a * x * ( x * x + 7 ) ;
```

7- حدد أولويات تنفيذ التعليمات التالية وقيمة x بعد تنفيذ كل منها :

```
a)     x = 7 + 3 * 6 / 2 – 1 ;
b)     x = 2 % 2 * 2 – 2 / 2 ;
c)     x = ( 3 * 9 * ( 3 + ( 9 * 3 /  3 ) )) ;
```

8- بفرض أن i = 6 , j = 3 , k = - 2

أوجد قيمة كل متغير في التعليمات التالية :

```
i + = 3 ;
k - = 5 ;
k * = 4 ;
j / = 2 ;
i + = j + 4 ;
i - = j - = 1 ;
++ j ;
k -- ;
```

```
h = j -- + i ;
h = -- j + i ;
h = j -- + ++ i ;
h = -- j + ++ i ;
```

9- اكتب العلاقات التالية بشكلها المكافئ بلغة C++ :

$$A = \frac{4x^2 + 10m}{5x/(3+m)} - 6m$$

$$b = 5L \leq 10m - 14$$

$$D = \frac{n.m^5}{5} + \frac{4m^2/10x}{5m + 3\sqrt{x}} - 6x$$

$$F = 5x \neq y + 4x$$

10- بفرض أنه لدينا القيم الصحيحة التالية :

result , n = 4 , m = 3 , l = 2

بيّن أولويات تنفيذ عمليات المعالجة ونتائجها لكل مما يلي:

1- result = 17 %n –l*m+6;
2- result = 17% (n-l)*m+6;
3- result = l * (7+(m+3)/2)-n;

11- تتبع تنفيذ التعليمات التالية وبين قيمة كل متغير بعد تنفيذ كل منها :

```
int x , y , z;
x =y = z = 6;
x * = y + = z - = 4;
```

الفصل الرابع

تعليمات الإدخال والإخراج

Input and output statements

في شرحنا للبرنامج البسيط في نهاية الفصل الثالث لم نحتج إلى التصريح عن أية بيانات متغيرة أو ثابتة أو لإدخال قيم ومن ثم معالجتها لأن كل ما يقوم به البرنامج هو طباعة العبارة Hello. Student . ولكن في الحياة العملية تجابهنا مشاكل عديدة نحتاج لإدخال بيانات من أجل معالجتها ومن ثم إخراجها للحصول على النتائج المطلوبة.

4-1- تعليمة التصريح عن البيانات (Declaration statements) :

لقد ذكرنا في الفقرة (3-2-4) من الفصل الثالث أنه لابدّ من التصريح عن المتغيرات والثوابت قبل استخدامها في البرنامج وذلك لإعطاء المترجم معلومات عن نوع (حجم) واسم المتغير.

تأخذ تعليمة التصريح الشكل العام التالي:

Data type List of variable_names;

حيث:

Data type : نوع بيانات المتغير. الجدول (3-1) حدد أنواع البيانات. وتتميز لغة C++ بأنها تقوم بحجز حيز في الذاكرة لتخزين قيمة المتغير حسب نوعه عند الإعلان (التصريح) عنه. ولكل نوع حجم معين من البيانات يستوعب قيماً عددية محدودة المجال.

List of variable _ names : لائحة بأسماء المتغيرات يفصل بين عناصرها الفاصلة العادية وهي أسماء مقبولة في اللغة ويسميها المبرمج.

كما يمكن إعطاء قيم ابتدائية Initialized للمتغيرات في تعليمة التصريح فتأخذ الشكل العام التالي:

```
Data type    variable name1 = initial value 1,
             variable name 2 = initial value 2;
```

مثال (1):

```
# include < iostream.h >
// A simple program to illustrate assignment :
main ()
{
        int n;
        n = 66;
        cout << " n = " << n << " \ n ";
        return 0;
}
```

الخرج هو :

```
n = 66
```

التعليمة ;int n تعلن أو تصرح عن متغير اسمه n من نوع الأعداد الصحيحة int . والتعليمة ;n = 66 تعين القيمة 66 للمتغير n .

>> cout تؤدي إلى طباعة العبارة = n المحصورة ضمن أداتي الحصر العلويتين ثم طباعة قيمة n وهي 66 في سطر واحد على الشاشة.

n \تؤدي كما ذكرنا في الفصل السابق إلى الانتقال لسطر جديد.

;return 0 : تدل على انتهاء تنفيذ البرنامج والعودة إلى نظام التشغيل.

في لغة ++C يمكن الإعلان عن المتغيرات والثوابت في أي موقع في البرنامج ولكن الشرط هو أن يتم ذلك قبل استخدام المتغير أو الثابت وذلك لكي يأخذ المترجم فكرة عن اسم ونوع المتغير أو الثابت كما تم ذكره سابقاً.

مثال (2) :

```
# include < iostream.h >
// This program illustrate variable declaration:
main ()
```

```
{
        int z,y;    // declares two variables   z and y.
        y = 5;     // assign  5 to  variable y.
        z = 10;   // assign 10 to variable z.
        B = 5;     // Error : undeclared variable
        int x = 50, B; // declares two variables and initiate x to 50.
        B = 12;
        cout << x << " , " << B << " \ n ";
        cout << z << " , " << y << " \ n ";
        return0;
}
```

الخرج هو :

```
50,12
10,5
```

التعليمة B = 5; خطأ ولايمكن تعيين القيمة 5 للمتغير B حيث لم يتم الإعلان عنه بعد.

2-4- إدخال البيانات باستخدام >> cin

الشكل العام لتعليمة الإدخال هي:

cin >> variable_name;

تبدأ التعليمة بالغرض cin وتلفظ (see – in) ويدعى بفرض مجرى الدخل القياسي (standard input stream) ثم يليه عامل الاستخلاص أو الإدخال >> (Extraction operator) ثم اسم المتغير المطلوب إدخال قيمة له.

هذه التعليمة تأمر الحاسوب بإدخال القيم من لوحة المفاتيح وإظهارها على الشاشة (نافذة المخرجات). ثم نقل القيم من الشاشة إلى مواقعها في الذاكرة وذلك بعد الانتهاء من الإدخال أي عندما يضغط المستخدم على مفتاح الإدخال (Enter)، تمهيداً لاستخدامها في عمليات المعالجة. لذلك عند تنفيذ البرنامج ووصول التنفيذ إلى التعليمة cin>> بظهر على شاشة الحاسوب المؤشر الوامض (cursor) عندها يقوم المستخدم

بكتابة القيمة المراد إدخالها باستخدام لوحة المفاتيح ثم يضغط على المفتاح Enter . عندها تخزن القيمة في الذاكرة في الموقع المسمى باسم المتغير الموجود في تعليمة الإدخال ، وفي أثناء المعالجة يتم استبدال المتغير أينما وجد في البرنامج بقيمته المخزنة.

variable_name : اسم المتغير الذي تم التصريح عنه بتعليمة سابقة.

تنتهي تعليمة الإدخال كبقية أوامر لغة ++C بالفاصلة المنقوطة (;) . يمكن أن تحتوي تعليمة الإدخال على أكثر من متغير مطلوب إدخال قيمهم ، عندها نفصل بين أسماء المتغيرات بالعامل << . عندها تأخذ تعليمة الإدخال الشكل التالي:

```
cin >> variable_nam1 >> variable_name2;
```

مثال :

```
cin >> x >> age;
```

حيث x و age أسماء لمتغيرات مطلوب إدخال قيم لها.

4-3- إخراج البيانات في برنامج ++C وتنسيقها:

4-3-1- تعليمة الإخراج >> cout

الشكل العام لتعليمة الإخراج هي :

```
cout << list;
```

حيث:

cout : هو غرض مجرى الخرج القياسي .

>> : عامل الإخراج Insertion operator

list : يمكن أن تأخذ بعضاً أو كلاً مما يلي :

1- اسم متغير يطلب إظهار قيمته على الشاشة.

2- اسم ثابت يطلب إظهار قيمته على الشاشة.

3- تعبير حسابي أو منطقي يُراد إظهار نتيجة الحساب على الشاشة.

4- قيمة رمزية أو تعليق يُراد إظهارهما على الشاشة شريطة وضعهما ضمن أداتي الحصر العلويتين (") quotation marks .

5- أحد رموز أو أوامر تنسيق المخرجات.

على أن يفصل بين الأنواع السابقة العامل >> وفي نهاية التعليمة لاننسى الفاصلة المنقوطة (;) التي تسمى أحياناً بمؤشر نهاية التعليمة (statement terminator).

مثال(3):

```
# include < iostream.h >
main ()
{
    char first, last;
    cout << " Enter your first and last initials:";
    cin >> first >> last;
    cout << " Hello," << first << " . "
         << last << " . ! \ n ";
    return 0;
}
```

الخرج هو :

```
Enter your first and last initials: J H ⏎
Hello, J.H.!
```

يبدأ البرنامج بتوجيه إلى المعالج المسبق بإضافة الملف الرأسي iostream.h إلى ملف البرنامج ثم اسم التابع الرئيسي لأي برنامج في لغة C++ وهو main () القوس الكبير الاستهلالي } يبدأ جسم التابع الرئيسي main () ، حيث جسم التابع main هو التعليمات التي تحل المسألة المطروحة وينتهي البرنامج بالقوس الكبير الختامي { .

جسم البرنامج يحوي أولاً تعليمة تصريح عن قيمتين محرفيتين char الأولى اسمها first والثانية last . تنتهي التعليمة طبعاً بالفاصلة المنقوطة.

يلي ذلك تعليمة الإظهار التي تظهر العبارة: Enter your first and last initiatials في نافذة المخرجات وهي الشاشة كتوضيح للمبرمج وتذكيره بما يجب أن يعمل، (لاتنسى تضمين أي عبارة تود إظهارها ضمن أداتي حصر علويتين "   ") .

أما تعليمة الإدخال لقيم كل من first و last فتنتهي أيضاً بالفاصلة المنقوطة وتحوي العامل << بين اسمي المتغيرين الحرفيين first و last .

تعليمة الإظهار التي تليها تنفذ بإظهار العبارة الموجودة ضمن أداتي حصر علويتين أولاً وهي Hello ، ثم يليها مباشرة إظهار قيمة المتغير first يليها القيمة ما بين أداتي الحصر وهي النقطة (.) ثم قيمة المتغير last ثم القيمة ما بين أداتي الحصر وهي (! .) . لاحظ أن العامل >> يفصل ما بين أوامر الخرج السابقة. الرمز n \ في آخر أمر الطباعة هو أمر الانتقال لسطر جديد.

التعليمة ;return 0 تنهي البرنامج بنجاح وتعود لبرنامج التشغيل كما ذكرنا ويختم جسم البرنامج بالقوس الكبير الختامي { الذي يدل على نهاية التابع () main .

4-3-2- تنسيق الخرج باستخدام رموز تتابع الهروب

:(Escape sequence characters):

رموز تتابع الهروب هي ثوابت رمزية ذات خط مائل \ (Back slash) . وهي شائعة الاستخدام ضمن تعليمة الإخراج >> cout وكل منها يؤدي وظيفة محددة تخبر مؤشر الطباعة عن مكان طباعة البيانات التالية لهذا الرمز ، وبشكل مناسب على أن توضع هذه الرموز ضمن أداتي حصر علويتين " " .

تعرفنا حتى الآن على الرمز (n\) الذي يؤدي إلى الانتقال إلى سطر جديد . فيما يلي الجدول (4-1) يبين بعض هذه الرموز شائعة الاستخدام ووظيفة كل منها.

جدول (4-1) رموز تتابعات الهروب

meaning الوظيفة		Escape sequence
تخبر مؤشر الطباعة بطباعة ما يلي من بيانات في سطر جديد	New line	" \ n "
تؤدي لطباعة البيانات التالية بنفس السطر ولكن بعد مسافة محددة قدرها tab	Horizantal tab	" \ t "
طباعة البيانات التالية قبل مسافة قدرها فراغ واحد من نفس السطر	Back space	" \ b "
حرف الإرجاع، يجبر المؤشر إلى العودة لبداية السطر الحالي	Carriage return	" \ r "
إخراج صوت الصافرة	Alert (bell)	" \ a "
يؤدي إلى طباعة أداتي الحصر العلوي "	Double quote	" \ " "
يؤدي إلى طباعة أداة حصر علوية واحدة ' .	Single quote	" \ ' "
يؤدي إلى طباعة الخط المائل \	Back slash	" \ \ "
يؤدي إلى طباعة علامة الاستفهام ؟ .	Question mark	" \ ? "
تقديم صفحة	Form feed	" \ f "
ترتيب عمودي	Vertical tab	" \ v "

مثال :

لو كان لدينا تعليمة الخرج التالية :

```
cout << " first line \ n second line ";
```

لكان الخرج سيظهر في سطرين منفصلين بسبب وجود الرمز (n \) كما يلي:

```
first line
second line
```

3-3-4- استخدام الأمر setw(N)

يستخدم الأمر setw(N) مع تعليمة الإخراج >> cout وقبل المعلومة المطلوب طباعتها بالشكل التالي:

```
Cout << setw(N) << y;
```

حيث :

N : عدد المواضع التي تحجز لطباعة قيمة y ضمنها.

يؤدي الأمر setw(N) إلى طباعة قيمة المتحول الذي يأتي بعده فقط، في حقل عرضه N موضع (رمز).

إذا كانت قيمة y المطلوب طباعتها تشغل مسافة أقل من N رمز ، عندها تتم طباعة هذه القيمة ضمن المسافة المخصصة لذلك مع تنسيقها بإزاحتها إلى اليمين. وإذا كانت هذه القيمة تشغل مسافة أكبر من المسافة المخصصة عندها تتم طباعة هذه القيمة كاملة من دون اقتطاع أي رمز منها.

مثال:

لو كان لدينا التعليمة التالية :

```
Cout << " mark = " << setw(4) << mark;
```

وكانت قيمة mark تساوي 81 . فإن الخرج الناتج من تنفيذ التعليمة السابقة هو:

1	2	3	4	5	6	7	8	9
m	a	r	k	=			8	1

حيث الحقول الأربعة ، من الحقل 6 وحتى الحقل 9 ، هي الحقول المحجوزة بالأمر setw(4) . وتم تنسيق الخرج إلى اليمين.

ملاحظة (1) : حتى يقوم الحاسوب بتشغيل الأمر setw(N) يجب إضافة الملف الرأسي لمعالجة المخرجات واسمه iomanip.h وهو اختصار لـ :

Input/output manipulation. Header file.

ويضاف هذا الملف في بداية البرنامج وقبل اسم التابع الرئيسي main () تماماً مثل إضافة الملف الرأسي iostream.h وذلك كما يلي:

```
# include < iomanip.h >
```

إذا أردنا تنسيق القيم المطبوعة بإزاحتها إلى يسار الحقل (N) عندها نستخدم استدعاء للأمر setiosflags(ios::left) ويدعى متحكم وسيطي بالمجرى parameterized stream manipulator وذلك كما يلي:

```
cout << " mark = "<< setw(4) << setiosflags(ios::left) << mark;
```

لو كانت قيمة mark هي أيضاً 81 . فإن الخرج الذي سيظهر هو كالتالي:

1	2	3	4	5	6	7	8	9	10
m	a	r	k	=	8	1			

حيث الحقول الأربعة ، من الحقل 6 وحتى الحقل 9 ، هي الحقول المحجوزة بالأمر setw(4) . وتم تنسيق الخرج 81 إلى يسار الحقل باستدعاء التابع setiosflags(ios::left) والذي يتطلب أيضاً تضميناً للملف الرأسي iomanip.h.

4-3-4- التحكم بشكل إخراج الأعداد الحقيقية:

يمكن التحكم بشكل صيغ الأعداد الحقيقية باستخدام المتحكمات الوسيطية بالمجرى .

مثال (4):

البرنامج التالي يوضح استخدام المتحكمات الوسيطية بالمجرى. يحسب البرنامج مقدار العمولة (commission) التي يتقاضاها مندوب مبيعات في شركة على مجمل

مبيعاته (total sales value) ، إذا علمت أن نسبة العمولة (commission rate) هي 20% . تحسب العمولة حسب العلاقة التالية :

قيمة العمولة = قيمة مجمل المبيعات * نسبة العمولة

```
# include < iostream.h >
# include < iomanip.h >
main ()
{
        double total _ sales _ value, rate = 0.2;
        double commission;
        cout << " enter total sale's value:";
        cin >> total _ sales _ value;
        commission = total_ sales _ value * rate;
        cout << " total commission is :"
              << setiosflags (ios:: fixed) << setiosflags(ios::showpoint)
              << setw(10) << setprecision (2)
              << commission << endl;
        return 0;
}
```

التعليمة الأولى هي تعليمة إدراج للملف الرأسي iostream.h إلى ملف البرنامج لتشغيل أوامر الدخل والخرج .

التعليمة الثانية هي تعليمة إدراج للملف الرأسي iomanip.h إلى ملف البرنامج والضروري لتشغيل الأوامر setw و setiosflags و setprecision وتدعى كما ذكرنا بمتحكمات وسيطية بالمجرى.

تتم طباعة قيمة العمولة والمصرح عنها من نوع double حقيقي مضاعف الدقة، حسب الصيغة التي يحددها الاستدعاء التالي لتثبيت الفاصلة وإظهارها setiosflags(ios::showpoint) و setiosflags(ios::fixed) مع تنسيق بالإزاحة نحو

اليمين ضمن مسافة مقدارها (10) مواضع مخصصة بالاستدعاء setw(10) وبدقة تصل إلى موضعين بعد الفاصلة العشرية بوساطة الاستدعاء setprecision(2) .

إذن من أجل تنسيق الخرج لقيم عددية حقيقية من النوع float أو double نخبر الحاسوب بما يلي:

1- عدد الخانات في الجزء العشري للعدد باستدعاء الأمر setprecision(N) حيث N عدد الخانات للجزء العشري يختاره المبرمج حسب التطبيق والحاجة.

2- استدعاء الأمر setiosflags(ios::fixed) لإظهار العدد بالشكل العشري العادي وليس بشكل الفاصلة العائمة.

3- استدعاء الأمر setiosflags(ios::showpoint) لإظهار أصفار في أقصى اليمين من الجزء الكسري استكمالاً لعدد الخانات في الجزء الكسري حسب الأمر setprecision(N) . وذلك قبل اسم المتحول المطلوب التحكم بشكل الخرج العددي له.

لاحظ وجود endl في نهاية تعليمة الإخراج (وهي اختصاراً لـ end line) والتي تسمى بمتحكم المجرى (stream manipulator) . يقوم المتحكم endl بإخراج نهاية سطر وإفراغ مخزن الخرج (output buffer) . هذا يعني أنه بالنسبة لبعض الأنظمة التي يتم فيها تجميع عمليات الخرج في مخزن حتى يصل مستواها إلى حد معين يستوجب طباعتها على الشاشة، فإن endl توقف عملية التجميع وتخرج كل ما تم تجميعه ليطبع على الشاشة فوراً من دون أي انتظار.

4-4- تمارين:

1- بفرض أن i=1 و j=2 و k=3 و m=2 ماذا تطبع كل من التعليمات التالية :

```
a)	cout << ( i = = 1 ) << endl;
b)	cout << ( j = = 3 ) << endl;
c)	cout << ( i > = 1 && j < 4 ) << endl;
```

```
d)      cout << ( m < = 99 && k < m ) << endl;
e)      cout << ( j > i  || k = = m ) << endl;
f)      cout << ( k+m<j || 3-j > = k ) << endl;
g)      cout << (!m) << endl;
h)      cout << ( ! (j-m) ) << endl;
i)      cout << ( ! (k > m) ) << endl;
```

التعليمات السابقة تطبع القيم التالية :

1 , 0 , 1 , 0 , 1 . 0 , 0 , 1 , 0 على التوالي، كل في سطر منفصل.

2) اكتب برنامجاً بلغة ++C لإيجاد ما يلي :

1- حساب مساحة ومحيط مربع علم طول ضلعه a قيمة صحيحة.

2- حساب مساحة ومحيط مستطيل علم طول ضلعاه l و w قيم صحيحة.

```
# include <iostream.h>
main()
{
    int a, sqarea, sqcircum;
    int l,w, rectarea, rectcircum;
    cout << " enter the length of the square side:";
    cin >> a;
    sqarea = a * a;
    sqcircum = 4*a;
    cout << " the square area is : " << sqarea << endl
          << " the square circumference is " << sqcircum << endl;
    cout << " enter the width and length of the rectangle:";
    cin >> w >> l;
    rectarea = w*l;
    rectcircum = 2*(l+w)
    cout << " the rectangle area is : " << rectarea << endl
          << " the rectangle circumference is:"
          << rectcircum << endl;
    return 0;
}
```

الخرج هو :

```
enter the length of the square side:7⏎
the square area is:49
the square circumference is:28
enter the width and length of the rectangle:3   8⏎
the rectangle area is:  24
the rectangle circumference is: 22
```

3) اكتب برنامجاً يحسب مساحة مثلث علم طول ضلعين منه والزاوية المحصورة بينهما ثم يحسب ويطبع طول الضلع الثالث

```
// program to find Area & third side of Triangle
# include <iostream.h>
# include <math.h>
main()
{
    const float pi = 3.14;
    float tetta, f, area;
    int a,b,c;
    cout << "enter two sides and angle:";
    cin >> a >> b >> tetta;
    f = tetta * pi/180;
    area = 0.5*a*b*sin(f);
    c = sqrt (a*a+pow (b,2)-2*a*b*cos(f));
    cout << " area of triangle is:" << area << endl;
    cout << " third side is: " << c << endl;
    return 0;
}
```

والخرج هو :

```
enter two sides and angle: 3  5  60
area of triangle is: 6.4932
third side is: 4
```

4) اكتب برنامجاً يحسب قيم المتغيرات c,b,a المعطاة لكل منهم بالعلاقة التالية :

$$a = x^2 + y^2$$
$$b = x + 3a$$
$$c = xy + a - b^2$$

حيث y , x قيم حقيقية معلومة:

```
# include <iostream.h>
# include < iomanip.h >
main()
{
    float a, b, c, x, y;
    cout << " enter two float values:"
    cin >> x >> y;
    a = x * x + y * y;
    b = x + 3 * a;
    c = x * y + a – b * b;
    cout << " a = "<< setw(8) << a << " \t b = " << b <<
          << " \t c = " << setw(10) << c << endl;
    return 0;
}
```

لو نفذ البرنامج من أجل x = 1.5 و y = 3.0 لكان الخرج هو :

```
enter two float values: 1.5   3.↵
a =   11.25         b = 35.25            c =  -1226.81
```

أما لو نفذ من أجل قيم x = 1000.0 و y = 0.25 لكان الخرج هو :

```
enter two float values: 1000.0    0.25↵
a =   1e+06         b =3.001e+06    c=-9.006e+12
```

5) اكتب برنامجاً يطلب إدخال قيمة حقيقية ويطبعها ثم يطبع القيمة الحقيقية (x=356.70) وناتج القسمة (3456789.70/2) وناتج الجداء (0.13*19.98) كل في سطر.

```
# include <iostream.h>
# include <iomanip.h>
main()
{
    double x;
    cout << "enter a floating number:";
    cin >> x;
    cout << " the value of x is: " << x;
    cout << " \n " << setw(10) << 356.7;
    cout << " \n " << 3456789.70/2;
    cout << " \n " << 0.13 * 19.98;
    return 0;
}
```

والخرج هو :

```
enter a floating numbere : 5.0
the value of x is : 5↵
   356.7
1.72839e + 06
2.5974
```

6) عدّل البرنامج السابق بحيث يطبع الأرقام بالشكل العشري العادي مع خانتين بعد الفاصلة العشرية وبإظهار أصفار في الجزء الكسري استكمالاً لعدد الخانات.

```
# include <iostream.h>
# include <iomanip.h>
main()
{
    cout << setprecision (2)
          << setiosflags (ios :: fixed)
           << setiosflags (ios :: showpoint);
    double x;
    cout << " enter a floating number:";
    cint >> x;
    cout << " the value of x is:"<< x;
```

```
     cout << " \n " << setw (10) << 356.70;
     cout << "\n " << 345789.70/2;
     cout << " \n " << 0.13 * 18.98;
     return 0;
}
```

والخرج يكون:

```
enter a floating number : 5↵
the value of x is: 5.00
   356.70
1728394.85
2.60
```

الفصل الخامس

تعليمات التحكم واتخاذ القرار

Control Statements & Decision Making

تعرضنا في الفصل الثاني لمصطلح الخوارزمية وارتباطه باللغات البرمجية وعرف على أنه مجموعة الخطوات المتسلسلة والمحددة التي تؤدي إلى حل قضية معينة والوصول إلى نتائجها.

البرامج التي استعرضناها حتى الآن استخدمت خطوات محددة متتالية تؤدي إلى النتيجة من دون الحاجة إلى تغيير سياق التنفيذ، وتنفذ خطوة خطوة حتى الوصول إلى النهاية وحسب ترتيب ورودها في نص البرنامج. يسمى أسلوب التنفيذ هذا بالتنفيذ التسلسلي (Sequential execution) سوف نناقش فيما يلي العديد من تعليمات C++ التي تساعد على تحديد التعليمة التالية في التنفيذ للتعليمة الحالية، والتي يمكن أن تكون في موضع ما من نص البرنامج، مختلف عن موضع التعليمة التالية لها مباشرة، ويسمى هذا الانتقال بنقل التحكم (Transfer of control) .

تمتلك لغة C++ ثلاثة أنواع من تعليمات بنى الاختيار (Selection structure).

5-1- تعليمة الشرط if (Conditional if statement):

وتسمى أيضاً بالبنية وحيدة الاختيار (Single selection structure) لأنها تسمح باختيار أو تجاهل فعل وحيد.

الشكل العام لتعليمة الشرط هو :

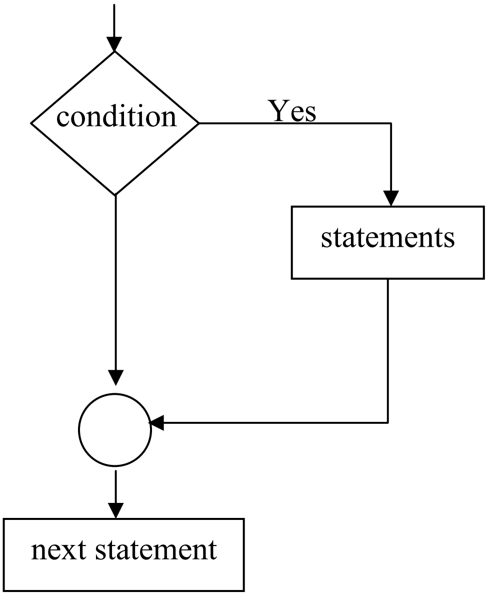
```
if ( condition )
  {
      statements;
  }
next statement;
```

حيث :

condition: هو تعبير مقارنة حسابية أو منطقية.

statements: تعليمة أو أكثر بلغة ++C ستنفذ إذا تحقق الشرط.

الشكل (5-1) يبين مخطط سير العمل للتعليمة if الشرطية.



الشكل (5-1) **مخطط سير العمل لتعليمة if الشرطية**

يقوم الحاسوب بفحص الشرط (condition) فإذا كانت قيمته محققة (True) أو لا تساوي الصفر، عند ذلك تنفذ التعليمة التي تلي الشرط if مباشرة، أو مجموعة التعليمات (statements) الموجودة ضمن القوسين { } وتسمى عندها بالتعليمة المركبة (compound statement) ، لاضرورة للأقواس { } إذا كان المطلوب تنفيذ تعليمة وحيدة لو تحقق الشرط.

أما إذا كانت قيمة الشرط غير محققة، أي خاطئة (False) ، أو تساوي الصفر فسيتم تجاهل تنفيذ statements ويتابع بالتنفيذ من التعليمة next statement والتي تلي التعليمة المركبة statements.

ملاحظة:

التعليمة المركبة هي مجموعة من التعليمات التي تنفذ كأنها تعليمة واحدة. ولغة C++ تميز التعليمة المركبة بوضعها ضمن القوسين { } .

المثال التالي يحوي التعليمة المركبة التي تقوم بالتبديل بين القيمتين الصحيحتين y , x باستخدام متغير ثالث صحيح temp .

```
{  int  temp = x;
           x = y;
           y = temp;
}
```

مثال (1): البرنامج التالي يدخل قيمتين صحيحتين first و Second ويظهر القيمة العظمى بينهما.

```
# include < iostream.h >
main ( )
{
        int first, second, max;
        cout << "Enter your first value:";
        cin >> first;
        cout << "Enter your second value:";
        cin >> second;
        max = first;
        if (second > max)
              max = second;
        cout << " The biggest value is:"
              << max << "\n";
        return 0;
}
```

يبدأ البرنامج بالتعليمة <include < iostream.h # التي ترشد المعالج المسبق بتضمين الملف الرأسي iostream.h إلى ملف البرنامج.

main( ): اسم التابع الأساسي في البرنامج والذي يبدأ بالقوس الكبير الاستهلالي {
يليها تعليمة التصريح عن ثلاثة متغيرات صحيحة max, second, first .

تعليمة الإظهار الأولى: cout << تخبر المبرمج أن عليه إدخال القيمة الأولى first

التعليمة: cin>> تطلب من المبرمج إدخال قيمة للمتغير الأول first

التعليمة: cout<< تعلم المبرمج بضرورة إدخال العدد الثاني second

التعليمة: max = first تعين القيمة الأولى وهي first لمتغير ثالث max.

التعليمة: if(second > max) تختبر هل القيمة الثانية second أكبر من max أي هل هي أكبر من القيمة الأولى first. إذا كان الجواب نعم ينتقل التنفيذ إلى التعليمة التالية مباشرة وهي تعيين قيمة العدد الثاني للمتغير max.

التعليمة: max = second تنفذ في حال كون القيمة الثانية second أكبر من max أي أكبر من القيمة الأولى first ، وبذا فإن max يحوي القيمة العظمى .

التعليمة: cout << تظهر العبارة "The biggest value is" يليها القيمة العظمى بين القيمتين المدخلتين ، وينتهي البرنامج بالقوس الكبير الختامي } .

لو تم إدخال القيمتين first=4 و second=10 لكان الشرط if(second >max) محققاً وعندها سينفذ الحاسب التعليمة max = second ويكون الخرج كما يلي:

```
Enter your first value:4 ⏎
Enter your second value:1⏎
The biggest value is 10
```

أما لو تم إدخال القيمتين first=8 و second=2 لكان الشرط if(second > max) غير محقق وعندئذ لاتنفذ التعليمة max = second وسيكون الخرج كما يلي :

```
Enter your first value : 8 ↵
Enter your second value : 2 ↵
The biggest value is 8
```

مثال (2) لنعيد كتابة البرنامج في المثال السابق ونجعل الحاسوب ينفذ تعليمة مركبة في حال تحقق الشرط if كالتالي:

```
# include < iostream.h >
main ()
{
        int first, second, max;
        cout << "Enter your first value:";
        cin >> first;
        cout << "Enter your second value:";
        cin >> second;
        max = first;
        if (max<second)
        {
              max = second;
              cout<<"The biggest value is the second one"
                   << endl;
        }
        cout<< "The maximum value is :"<< max << "\n";
        return 0;
}
```

لو أدخلت القيمتان first=4 و second=10 لكان الشرط if(max > second) محققاً وعندها سينفذ الحاسب التعليمة max = second ويكون الخرج كما يلي:

```
Enter your first value:4 ↵
Enter your second value:1 ↵
The biggest value is the second one
The maximum value is  10
```

أما لو تم إدخال القيمتين first=8 و second=2 لكان الشرط if(max < second) غير محقق وعندئذ سيتم تجاهل التعليمة المركبة بعد if والتي تضم تعليمتي تعيين وطباعة . ويكون الخرج كما يلي :

```
Enter your first value : 8 ↵
Enter your second value : 2 ↵
The maximum value is 10
```

2-5- تعليمة الشرط if...else (The if/else selection statement):

وتسمى بالبنية مضاعفة الاختيار Double selection structure أي أنها تسمح بالاختيار بين فعلين مختلفين.

الشكل العام للتعليمة هو كما يلي:

```
if (condition)
    { statements1;
    }
else
    { statements2;
    }
next statement;
```

حيث :

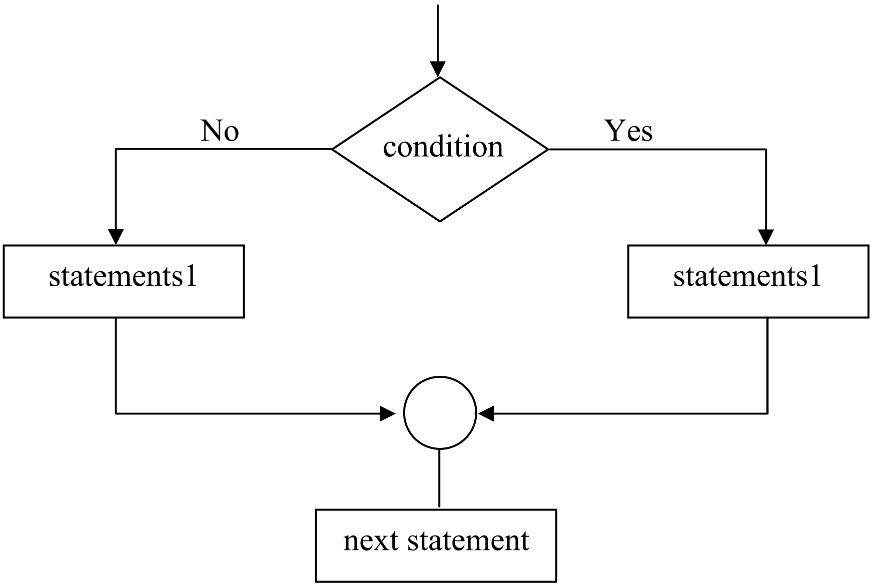
condition : هو تعبير مقارنة حسابية أو منطقية.

statements1 : تعليمة أو أكثر ستنفذ لو تحقق الشرط ثم ينتقل التنفيذ إلى next statement.

statements2: تعليمة أو أكثر ستنفذ لو لم يتحقق الشرط ومن ثم يتابع بتنفيذ التعليمة التالية next statement.

لاضرورة لوجود الأقواس الكبيرة { } إذا كانت أي من statements1 أو statements2 عبارة عن تعليمة وحيدة.

الشكل (2-5) يبين مخطط سير العمل للتعليمة if...else الشرطية:



الشكل (2-5) **مخطط سير العمل للتعليمة** if...else **الشرطية**

يقوم الحاسوب هنا أيضاً بفحص الشرط condition فإذا كانت قيمته محققة (true) ، أو لاتساوي الصفر، عند ذلك تنفذ التعليمات statements1 وينتقل التنفيذ إلى التعليمة التالية next statement. وتهمل التعليمات statements2.

أما إذا لم يتحقق الشرط، أو كان يساوي الصفر عندها تهمل التعلميات statements1 وتنفذ التعليمات statements2 ويتابع بتنفيذ التعليمة التالية next statements إذا كانت التعليمات Statements1 أو Statements2 عبارة عن تعليمة وحيدة عندئذ تحذف الأقواس الكبيرة { }.

يمكن إعادة كتابة البرنامج السابق كما يلي:

```
# include < iostream.h>
main()
{       int first, second, max;
        cout << "Enter your first value :";
        cin >> first;
```

```
    cout << "Enter your second value:";
    cin >> second;
    if (first > second)
        max = first;
    else
        max = second;
    cout << " The biggest value is " << max << endl;
    return 0;
}
```

لو أدخلنا أولاً القيمتين first=4 و second=10 كما في البرنامج السابق لكان الشرط if(first > second) غير محقق (false) وعندها لاينفذ الحاسب التعليمة max=first وينتقل التنفيذ للتعليمة max = second ثم يتابع بتنفيذ التعليمة التالية وهي تعليمة الإخراج وسيكون خرج البرنامج كالتالي:

```
Enter your first value:4 ↵
Enter your second value:10↵
The biggest value is 10
```

أما لو أدخلنا القيمتين first=8 و second=2 لكان الشرط if(first > second) محققاً (true) ونفذ الحاسب التعليمة التالية لتعليمة الشرط وهي max = first وأهمل التعليمة max = second أي لم ينفذها ثم ينتقل لينفذ التعليمة التالية وهي تعليمة الأخراج. ويكون خرج البرنامج كما يلي:

```
Enter your first value:8 ↵
Enter your second value:2 ↵
The biggest value is 8
```

3-5- تعليمة التعبير الشرطي (conditional expression statement):

تمكن لغة C++ بكتابة تعليمة الشرط if/else بطريقة مختصرة باستعمال العملية الشرطية (?:) conditional operator. وتعد العملية الشرطية عملية ثلاثية لأنها تأخذ

ثلاثة معاملات. ونسمي التعبير الذي يحوي العملية ومعاملاتها بالتعبير الشرطي ويأخذ الشكل التالي:

```
expression1 ? expression2: expression3;
```

حيث يشكل المعامل الأول expression1 الشرط المختبر في حين يشكل المعامل الثاني expression2 الجزء الذي يتم تنفيذه في حال تحقق الشرط المختبر. ويشكل المعامل الثالث expression3 الجزء الذي ينفذ في حال عدم تحقق الشرط المختبر.

مثال: لدينا قيمتان صحيحتان a , b والمطلوب تحديد القيمة الصغرى منهما للمتحول الصحيح min. يمكن الحصول على ذلك بكتابة التعليمة التالية:

```
min = a < b ? a : b;
```

حيث تتم مقارنة a مع b أولاً وإذا كانت a أقل من b أي الشرط a < b محققاً عندها تعين قيمة المتحول a للمتحول min وإلاّ فإنه سوف تعين قيمة المتحول b للمتحول min.

يمكن أن تكون نتيجة التعبير الشرطي تعليمة ما يجب تنفيذها . على سبيل المثال يمكن كتابة المثال السابق كالتالي:

```
min = a < b ? cout << min = a : cout << min = b;
```

4-5- الشروط المركبة :

تحتاج بعض المسائل إلى ضرورة تقويم أكثر من شرط واحد قبل تنفيذ مهمة ما. مثلاً يمكن جمع الشرط a % b و x > y معاً ليكونا شروطاً مركبة تستخدم عوامل المقارنة المنطقية لهذا الغرض (انظر الجدول رقم (3-4) من الفصل الثالث) .

مثال (3): البرنامج التالي يدخل ثلاثة قيم صحيحة ويطبع القيمة العظمى من بين تلك القيم:

```
# include < iostream.h>
main()
{ // declare three int variables
        int a , b , c;
        cout << "enter three integer values:";
        cin >> a >> b >> c;
        // compound if statement
        // to find the biggest of three variables
        if (a > = b && a > = c )
              cout << "the largest value is: " << a;
        if (b >> a && b > = c )
              cout << " the largest value is: " << b;
        if (c > = a && c > = b )
              cout << " the largest value is: " << c;
        return 0;
}
```

رأينا أن التعابير المنطقية (Boolen) تقدر بقيم صحيحة int، فالصفر يعني أن الشرط غير صحيح false وأية قيمة غير الصفر تعني أنه محقق true. أي أن كل الأعداد الصحيحة التي لاتساوي الصفر تعني أن الشرط صحيح.

التعليمتان التاليتان مثلاً :

```
if(n)
 cout << " n is not zero ";
```

سوف تطبع على الشاشة العبارة n is not zero عندما تكون n ليست صفراً لأن ذلك يجعل التعبير البولوني صحيحاً. مثال آخر:

```
if(n%d)
cout << " n is not a multiple of d";
```

هذه العملية سوف تطبع العبارة "n ليست من مضاعفات d "n is not a multiple of d" إذا كان الشرط محققاً أي لايساوي الصفر، أي أن n لاتقبل القسمة على d بدون باق . حيث أن n % d هو الباقي من ناتج القسمة.

في بعض الأحيان يمكن أن تحدث أخطاء رغم أن التركيب اللغوي صحيح ولايمكن للمترجم compiler أن يكتشفه . وهذا من أسوأ أنواع الأخطاء لأنه لايوجد دليل واضح يدل على وجود الخطأ.

مثال : لو كتبنا :

```
if (x > = y > = z)
cout << "max = " << x;     // Error!
```

فمن الواضح أن المبرمج يود كتابة:

```
if (x > = y && y > = z )
        cout << " max = " << x ; // ok.
```

لنفرض في الحالة الأولى أن كلاً من x , y تساوي صفراً و z تساوي الواحد، التعبير (x > = y > = z) يحسب من اليسار إلى اليمين .

الجزء الأول x > = y صحيح ويأخذ القيمة العددية 1 . عند ذلك يقارن بـ z وبما أن قيمة z تساوي 1 والقيمة العددية للمقارنة الأولى تساوي الواحد فإن قيمة التعبير الكلي السابق تكون صحيحة مع أنها فعلاً خاطئة.

هنالك خطأ آخر يمكن أن يقع به المبرمجون عند استخدام علاقة التساوي المفردة (=) والمستخدمة للتعيين بدلاً من علامة التساوي المزدوجة على سبيل المثال إذا كتبنا :

```
if(x = 0)      //   Error!
cout << " x = 0 " ;
```

والمقصود كتابة التالي:

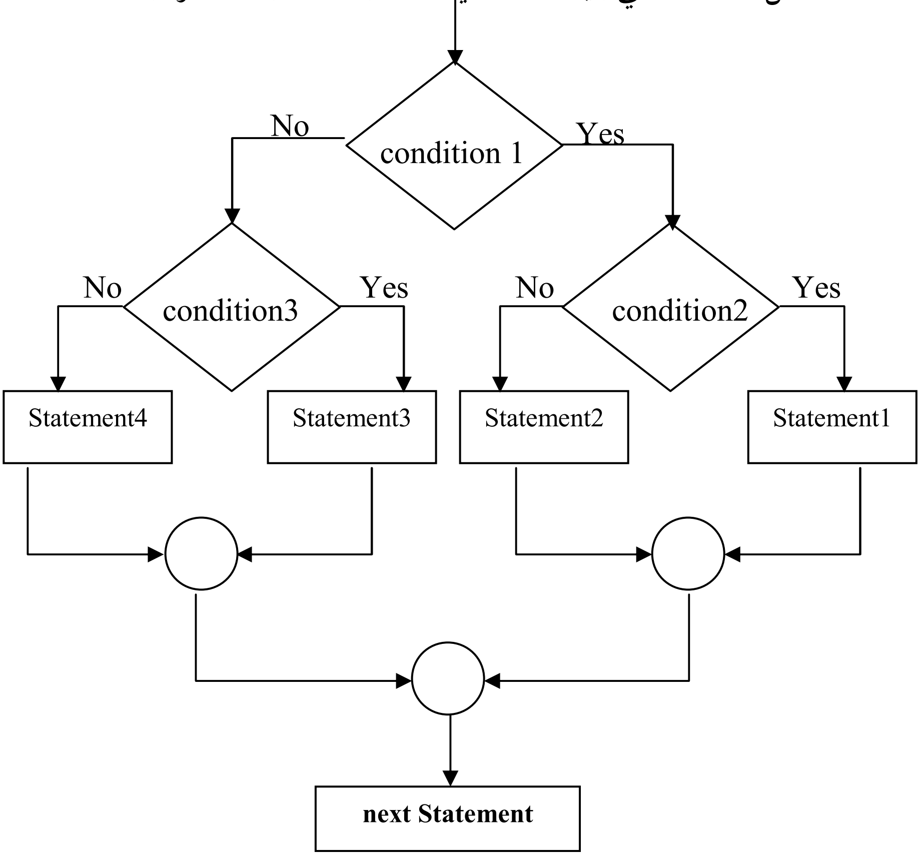
```
if(x = = 0)     // Ok
cout << " x = 0 " ;
```

التعليمة الخاطئة سوف تعين القيمة صفر للمتحول x وهذا يعني أن تعبير المقارنة غير صحيح لأن له قيمة صفرية. وبذلك لاتنفذ تعليمة الطباعة حتى ولو كانت قيمة x الأصلية صفرية. أما إذا كانت لاتساوي الصفر فإنها سوف تأخذ القيمة صفر على كل حال.

5-5- الشروط المتداخلة Nested if statement

ذكرنا في الفقرة السابقة أن بعض المسائل يحتاج حلها إلى استخدام شروط مركبة وهنالك مسائل أخرى تستدعي وجود تعليمة شرطية داخل تعليمة شرطية أخرى.

الشكل (5-3) التالي يبين مخطط سير العمليات لتعليمات شرطية متداخلة.



الشكل (5-3) **مخطط سير العمليات لتعليمات شرطية متداخلة**

من المخطط نلاحظ وجود أربعة مسارات للتنفيذ.

المسار الأول : يتم اختبار الشرط condition1 وإذا كان محققاً ينتقل لاختبار الشرط (condition2) وإذا كان هو أيضاً محققاً ينفذ التعليمة statement1 ثم ينتقل لينفذ التعليمة التالية next statement.

المسار الثاني : إذا كان الشرط (condition1) محققاً والشرط (condition2) غير محقق عندها تنفذ التعليمة statement2 ثم التعليمة التالية next statement.

المسار الثالث : إذا كان الشرط (condition1) غير محقق والشرط (condition3) محققاً عندها تنفذ التعليمة statemets3 ثم التعليمة التالية next statement.

المسار الرابع : إذا كان الشرط (condition1) غير محقق والشرط (condition3) غير محقق أيضاً، عندها تنفذ التعليمة statement4 ثم التعليمة التالية next statement.

أي من المسارات الأربعة يمكن أن ينفذ أولاً وليس بالضرورة أن تأخذ المسارات الترتيب السابق وذلك عائد لنتيجة تحقق أو عدم تحقق الشروط المختبرة.

يمكن إهمال المسار الذي لايطلب تنفيذ أية مهمة في حال تحقق شروطه.

الصيغة العامة لمخطط سير العمليات السابق هو :

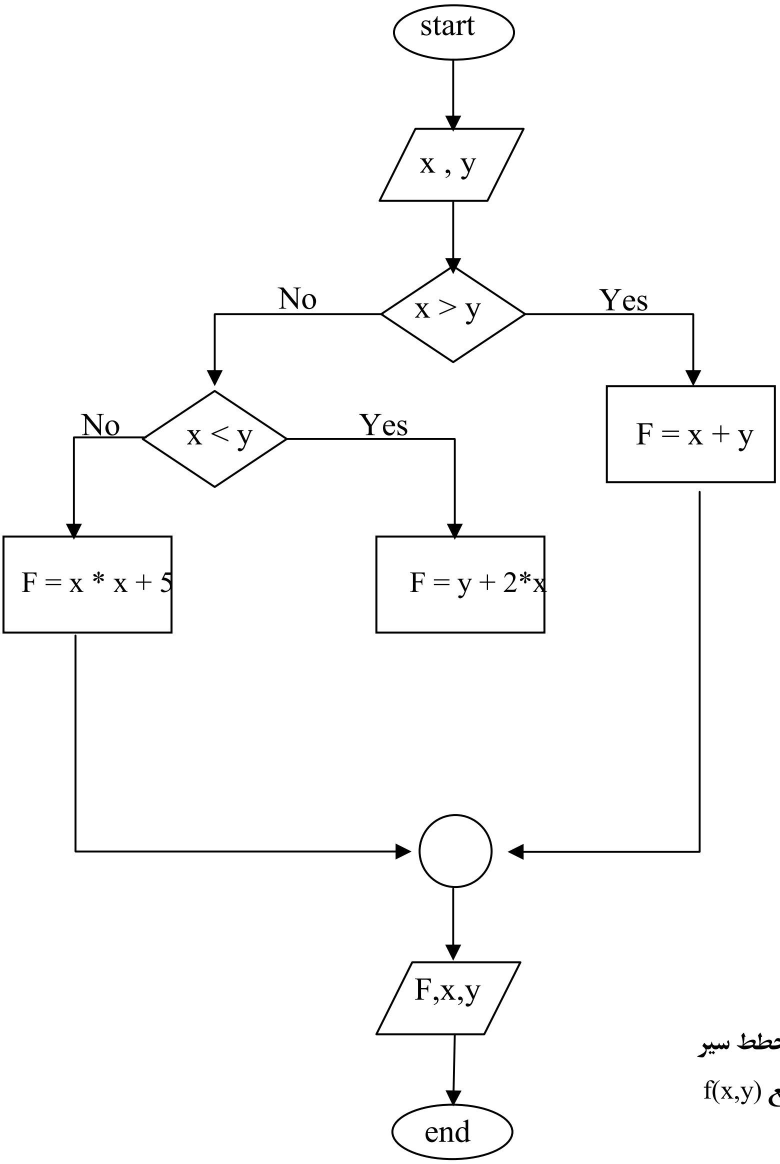
```
if (condition1)
    {
        if (condition2)
            {
                Statements1;
            }
        else
            {
                Statements2;
            }
    }
else
    {
        If (condition3)
            {
                Statements3;
            }
        else
            {
                Statements4;
            }
       }
next statement
```

مثال (4) : ارسم مخطط سير العمليات والبرنامج الذي يحسب قيمة التابع f(x,y) حسب التعابير والقيود التالية:

$$f(x,y)=\begin{cases} x+y & ; x>y \\ x^2+5 & ; x=y \\ y+2x & ; x<y \end{cases}$$

ثم يطبع قيم كل من x و y و f .

أولاً – الشكل (5-4) التالي يبين مخطط سير العمليات للتابع f(x,y) :



الشكل(5-4) **مخطط سير العمليات للتابع** f(x,y)

ثانياً – البرنامج

```
# include < iostream.h>
main ()
{
        int x, y, F;
        cout << " \n enter x value :";
        cin >> x ;
        cout << " \n enter y value:";
        cin >> y;
        if (x > y)
              F = x + y;
        else
              {
                   if (x < y)
                        F = y + 2 * x;
                   else
                        F = x * x + 5;
              }
        cout << " \n F = " << F << " \t when x = " << x
              << " \t and y = " << y << endl;
        return 0;
}
```

البرنامج السابق يصرح عن ثلاثة متغيرات صحيحة F , y , x ثم يطلب إدخال قيمة كل من y , x وينفذ الاختبار (x > y) فإذا كان محققاً ينفذ التعليمة F = x + y ثم ينتقل ليطبع العبارات والقيم لكل من y , x , F نتيجة تنفيذه تعليمة الطباعة الأخيرة.

أما إذا لم يتحقق الشرط (x > y) عندها ينتقل التنفيذ لبعد else ويختبر الشرط الثاني (x<y) وإذا كان محققاً ينفذ الحاسوب تعليمة التعيين F = y + 2 * x; وينتقل

لتنفيذ تعليمة الطباعة الأخيرة. أما إذا كان الشرط الثاني (x < y) غير محقق فإنه ينفذ تعليمة التعيين ;F = x * x + 5 ثم ينتقل لينفذ تعليمة الطباعة الأخيرة.

نلاحظ في المثال السابق استخدام الأقواس الكبيرة { } لتوضيح عملية التلازم بين if و else وتحويل كل الكود بينهما إلى تعليمة واحدة بحيث يلحق عبارة else بتعليمة if التي تسبقها كما هو مطلوب.

لندرس المثال التالي :

```
If ( a < b )     // if a less than b
        If ( a > c )    // and if a greater than c
              a = c;    // then set a to the value c
```

كتابة التعليمات بالشكل المتداخل السابق يوضح أن المطلوب التأكد من أن (a<b) و (a>c) لتعيين قيمة c للمتغير a . ولكن لو تداخلت التعليمات بشكل أكبر لأصبح من غير الواضح الترابط ما بين الشروط وضاع منطق تعيين تعليمات else إلى تعليمة if المناسبة.

مثال (5) البرنامج التالي يوضح مشكلة تداخل تعليمات المقارنة وعدم وضوح منطق تعيين تعليمات else إلى تعليمة if المناسبة:

```
//  a program demonstrates why braces
//  are important in nested if statements
# include < iostream.h>
main ()
{
        int a;
        cout << " Enter a number less than 8 or
                     greater than 50 :";
        cin >> a;
        if (a > 8)
              if (a > 50)
                   cout << " \n a is more than 50,";
7       else
8               cout << " \n a is less than 8";
9       return 0;
}
```

تم ترقيم أسطر البرنامج للشرح فقط ويجب عدم طباعتها عند التنفيذ.

يطلب البرنامج وحسب تعليمة الإظهار في السطر 2 إدخال قيمة لـ a أقل من 8 أو أكبر من 50 ويطبع العبارة المناسبة.

التعليمة في السطر (4) يتم اختبار قيمة a المدخلة بالتعليمة cin >> a ولو كانت نتيجة الاختبار صحيحة (أي a أكبر من 8) تنفذ التعليمة التالية وهي اختبار قيمة a مرة أخرى هل هي أكبر من 50 . وإذا كانت نتيجة الاختبار صحيحة طبعت العبارة الموجودة في تعليمة الخرج الموجودة في السطر 6 . أما لو كانت تعليمة if الثانية غير صحيحة لكان الخرج حسب التعليمة في السطر 8 وهذا خاطئ.

البرنامج السابق يعطي خرجاً صحيحاً إذا أدخلت قيمة لـ a أكبر من 50 ، أما لو أدخلت قيمة لـ a أقل من 50 وأكبر من 8 لكان الخرج خاطئاً وهو الخرج الناتج عن التعليمة في السطر 8 .

في حال كون قيمة a المدخلة أقل أو تساوي 8 فلن يكون هنالك خرج لأنه سيتحول التنفيذ إلى التعليمة في السطر 9 . وهي نهاية البرنامج.

نعيد كتابة البرنامج السابق بإدخال الأقواس الكبيرة { } في المكان المناسب ويصبح:

```
# include < iostream.h >
main ()
{
1    int a;
2    cout << " Enter a number less than 8
                   or greater than 50:";
3    cin >> a;
4    if (a > 8)
```

```
        {
5               if ( a > 50 )
6               cout << " \n a is more than 50,";
        }
7   else
8       cout << " \n a is less than 8";
9   return 0;
}
```

تعليمة الاختبار الأولى في السطر (4) إذا كانت محققة سيتم تنفيذ ما بين القوسين الكبيرين { } في السطرين 5 و 6 . أما إذا لم يتحقق الشرط في السطر 4 فإن التعليمة في السطر 8 ستنفذ والتي تأتي بعد else. إذن وجود القوسين الكبيرين جعل else تتبع if في السطر 4 بدلاً من إلحاقها بتعليمة if في السطر 5.

الآن إذا كانت a أقل من 8 سيكون الخرج :

```
a is less than 8
```

وهو ناتج تنفيذ التعليمة في السطر 8

أما إذا كانت قيمة a أكبر من 8 وأكبر من 50 سيكون الخرج :

```
a is more than 50
```

وهو ناتج تنفيذ التعليمة في السطر 6.

إذا كانت a أكبر من 8 ولكنها أقل من 50 فلن يكون هنالك خرج وسينتقل التنفيذ من التعليمة في السطر 6 إلى التعليمة في السطر 9 وهي نهاية البرنامج.

مثال (6): اكتب برنامجاً لحساب عمولة بائع على مجموع مبيعاته إذا علمت أن نسبة العمولة تساوي 2% إذا كان مجموع مبيعاته أقل من 500 ل.س، وتساوي 5% إذا كان مجموع مبيعاته أكبر أو يساوي 500 ل.س وأقل أو يساوي 5000 ل.س، وتساوي 8% إذا كان مجموع مبيعاته أكبر من 5000 ل.س مع العلم أن :

العمولة = مجموع المبيعات * نسبة العمولة المناسبة لمجموعة المبيعات.

البرنامج:

```
# include < iostream.h >
main ()
{
        int sale, commission;
        cout << " \n Enter amount of sale: ";
        cin >> sale;
        if (sale < 500)
              commission = 0.02 * sale;
        else
        {
              if (sale < 5000)
                   commission = 0.05 * sale;
              else
                   commission = 0.08 * sale;
        }
        cout << " \n commission = " <<commission << endl;
   return 0;
}
```

مثال (7): اكتب برنامجاً لحساب جذور معادلة من الدرجة الثانية والتي تأخذ الشكل :

$$ax^2 + bx + c = 0$$

وطباعتها مع العلم أنه : إذا كانت a تساوي الصفر عندئذ $x = c/b$.

أما إذا كانت a لاتساوي الصفر عندئذ يحسب المميز D وفق العلاقة:

$$D = b^2 - 4ac$$

وتحسب قيمة الجذور كما يلي :

1- إذا كانت $D >= 0$ عندئذ :

$$x_1 = \frac{-b + \sqrt{D}}{2a}$$

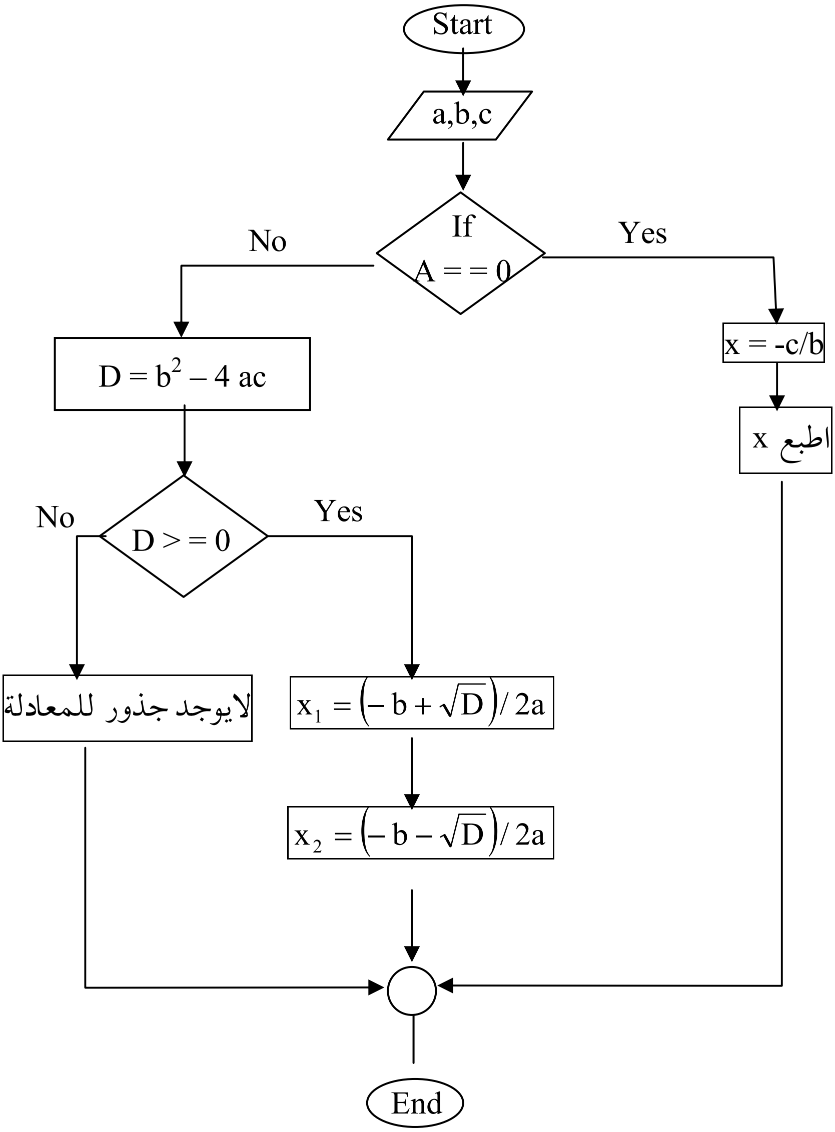
$$x_2 = \frac{-b - \sqrt{D}}{2a}$$

2- إذا كانت $D < 0$ عندئذ لايوجد جذور حقيقية للمعادلة.

أولاً - البرنامج الذي يحقق ذلك هو :

```
# include < iostream.h >
# include < math.h >
main ()
{
        float a,b,c,x,x1,x2,D;
        cout << " \n enter a,b,c: ";
        cin >> a >> b >> c;
        if ( a = = 0)
        {
              x = -c/b;
              cout << " \n x = " << x;
        }
        else
        {
              D = b * b – 4 * a * c;
              cout << " \n D = " << D;
              if ( D > = 0 )
                   {
                       x1=( - b + sqrt (D))/(2*a);
                       x2=(-b-sqrt(D))/(2*a);
                       cout << " \n root 1 = " << x1;
                       cout << " \n root 2 = " << x2;
                   }
              else
                cout<<"\n no real root";
        }
    return 0;
}
```

ثانياً – الشكل (5-5) يبين مخطط سير العمل لبرنامج حساب جذور معادلة من الدرجة الثانية .



الشكل (5-5) **مخطط سير العمل لبرنامج حساب جذور معادلة من الدرجة الثانية**

5-6- تعليمة الاختيار المتعدد

switch (The switch multiple selection structure):

تتطلب بعض المسائل سلسلة من القرارات التي تتعلق بنتائج متعددة لفحص قيمة متحول أو تعبير ما . حيث تؤدي كل نتيجة من هذه النتائج إلى القيام بفعل مختلف عن غيرها.

توفر لغة ++C البنية switch من أجل التعامل مع حالات اتخاذ القرار المتعلقة بعدة اختيارات.

الشكل العام لبنية الاختيار المتعدد switch كما يلي:

```
switch (expression)
{
        case  case _ value 1 :    statements 1;
                                    break;
        case  case _ value 2 :    statements 2;
                                    break;
                      :
                      :
        case  case _ value n :    statements n;  break;
        default :   statements;
}
```

حيث:

switch : كلمة مفتاحية تبدأ بها بنية الاختيار المتعدد.

expression : تعبير مقبول في لغة ++C أو اسم متحول أو قيمة صحيحة تقارن قيمته مع قيم الحالات (case values) المذكورة.

case : حالة ذات قيمة محددة case value . إذا تساوت معها قيمة التعبير المختبر عندها تنفذ التعليمات المتعلقة بهذه الحالة ثم ينفذ الأمر break والموجود في نهاية تعليمات كل حالة، والذي ينقل التنفيذ إلى خارج بنية switch .

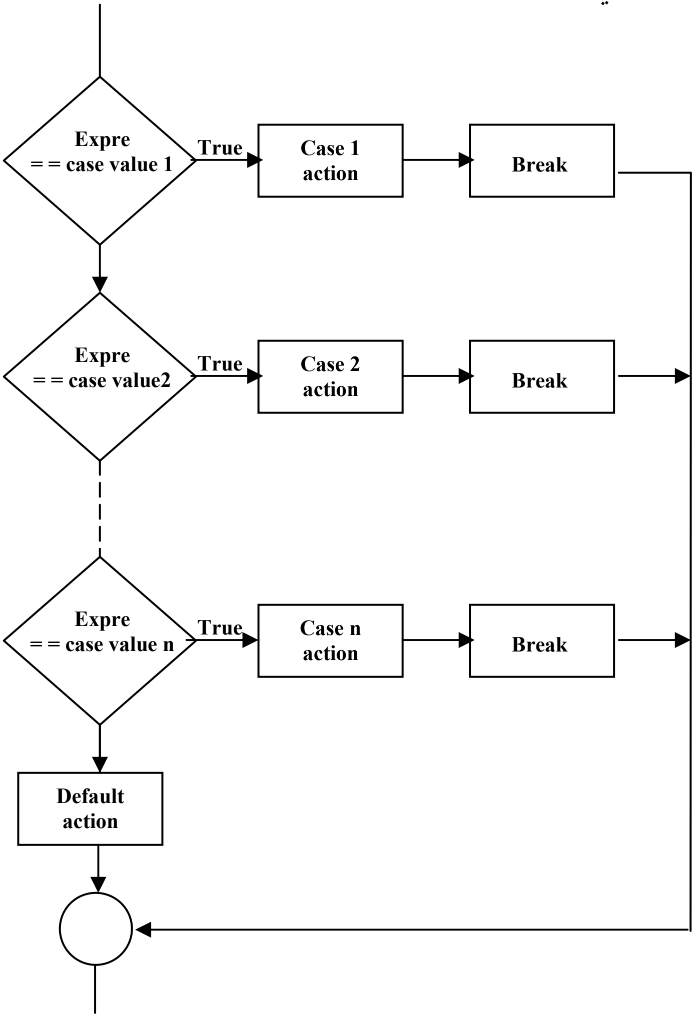
default : هو الموضع الذي ينتقل له التنفيذ فيما لو لم يكن هنالك أية قيمة مساوية لقيمة التعبير في أي من الـ case_values المذكورة في بنية switch وهي اختيارية.

ملاحظة (1) : المقارنة هنا تبحث عن المساواة فقط ولايمكن استعمال عوامل المقارنة الحسابية الأخرى أو المنطقية Boolen.

ملاحظة (2) : من المفيد والمهم أن تعرف أن إلغاء التعليمة break في آخر تعليمات أية حالة case سيحول التنفيذ إلى الحالة التالية، وقد يكون هذا الأمر ضرورياً أحياناً ولكنه مرفوض بشكل عام ويعتبر خطأً.

كما أنه من الضروري أن تكون قيم الحالات مختلفة فليس من المنطق استعمال ثابتين مختلفين من أجل حالة واحدة.

الشكل (5-6) التالي يبين مخطط سير العمليات لآلية تنفيذ بنية الاختيار المتعدد:



الشكل (5-6) مخطط سير العمل لآلية تنفيذ بنية الاختيار المتعدد

يمكن ترتيب الحالات cases ضمن البنية switch كما نريد لأن تنفيذها لايتعلق بترتيب كتابتها ولكن جرت العادة على وضع الحالة default في نهاية البنية. وعندها لايوجد ضرورة لكتابة تعليمة break.

مثال (8):

اكتب برنامجاً لحساب الراتب الشهري لموظف بعد إضافة علاوة تضاف لراتبه حسب الدرجة التقديرية التي تعطى له بحيث إن :

الراتب الشهري = الراتب الأساسي + مقدار العلاوة

إذا علمت أنه :

1- إذا كانت درجة تقدير الموظف 0 لايمنح أية علاوة.

2- إذا كانت درجة تقدير الموظف 1 يمنح علاوة قدرها 30%.

3- إذا كانت درجة تقدير الموظف 2 يمنح علاوة قدرها 40%.

4- إذا كانت درجة تقدير الموظف 3 يمنح علاوة قدرها 70%.

5- فيما عدا ذلك يأخذ علاوة قدرها 80%.

ثم يطبع في الخرج ، درجة تقدير الموظف grade ، وراتبه الأساسي basic _ seclary ، ومقدار العلاوة bouns والراتب الشهري المقبوض Salary.

البرنامج التالي يحقق ذلك:

```
# include < iostream.h >
main ()
{
        float basic_salary, bouns, salary;
        int grade;
        cout << " enter basic salary : " ;
        cin >> basic_salary;
        cout << " enter grade : " ;
start : cin >> grade;
        if (grade < 0)
        goto start;
        switch (grade)
        {
            case 0 :   bouns = 0;
                         break;
            case 1 :     bouns = basic_salary * 0.3;
                         break;
            case 2 :     bouns = basic_salary * 0.4;
                         break;
            case 3 :     bouns = basic_salary * 0.7;
                         break;
            default :    bouns = basic_salary * 0.8;
                         break;
        }
        salary = bouns + basic_salary;
        cout << " basic_salary = " << basic_salary
            << " \n grade = " << grade
            << " \n bouns = " << bouns
            << " \n salary = " << salary << endl;
          return 0;
}
```

مثال (9): المطلوب كتابة برنامج يدخل قيمة ثم يطبع العبارات الموافقة للقيمة المدخلة.

البرنامج هو :

```
# include < iostream.h >
main ()
{
  int n;
  cout << " enter a value between one and four : ";
  cin >> n;
  switch (n)
    {
    case 1 :   cout << " one \n "; break;
    case 2 :
    case 3 :   cout << " two or three \n "; break;
    case 4 :   cout << " four  \n "; break;
```

```
     default : cout << " default \n "; break;
     }
  cout << " end of switch. \n" ;
 return 0;
}
```

لو كانت القيمة المدخلة لـ n = 3 لكان الخرج :

```
two or three
end of switch
```

5-7- تمارين

1- اكتب برنامجاً يدخل عددين صحيحين ويختبر إذا كان العدد الأول n يقبل القسمة على العدد الثاني m وإذا تحقق ذلك يطبع العبارة "is divisible by"

```
# include < iostream.h >
main()
{
     int n, m;
     cout << "enter two integers:";
     cin >> n >> m;
     if (n%m = = 0)
     cout << n << "is divisible by" << m << endl;
     return 0;
}
```

الخرج هو:

```
enter two integers: 24 6↵
24 is divisible by 6
```

2- أعد كتابة البرنامج السابق واجعله يطبع العبارة "is not divisible by" في حال عدم تحقق الشرط.

تأكد أن العدد الثاني لايساوي الصفر.

```
# include < iostream.h >
main()
```

```
{
    int n, m;
    cout << "enter two integers:";
    cin >> n >> m;
    if ((m! = 0) && (n% m = = 0))
    cout << n << "is divisible by" << m << endl;
    else
    cout << n << "is not divisible by" << m << endl;
    return 0;
}
```

الخرج هو:

```
enter two integers: 29 5⏎
29 is not divisible by 5
```

3- اكتب برنامجاً يدخل علامة الطالب في مادة البرمجة في المتحول الصحيح mark . وإذا كانت العلامة أكبر أو مساوية لـ 90 يطبع الحرف A وإذا كانت العلامة بين 80 و 89 يطبع الحرف B . وإذا كانت العلامة بين 70 و 79 يطبع الحرف c . وإذا كانت العلامة بين 60 و 69 يطبع الحرف D . وإذا كانت العلامة أصغر من 50 يطبع الحرف F .

```
# include < iostream.h >
main()
{
    int mark;
    cout << " enter student mark:";
    cin >> mark;
    if (mark > = 90)
       cout << 'A' << endl;
    else  if (mark > = 80)
              cout << 'B' << endl;
           else  if (mark > = 70)
                 cout << 'C' << endl;
```

```
                    else  if (mark > = 60)
                              cout << 'D' << endl;
                            else
                               cout << 'F' << endl;
    return 0;
}
```

والخرج هو :

```
enter student mark : 72⏎
C
```

4- اكتب برنامجاً يدخل عددين صحيحين m , n وأحد رموز العمليات الحسابية، ثم ينفذ العملية الحسابية المدخل رمزها على العددين ويطبع الناتج.

```
# include < iostream.h >
main()
{
    int m,n;
    char ch;
    cout << "enter two integers:";
    cin >> m >> n;
    cout << "enter the operation:";
    cin >> ch;
    switch (ch)
    {
      case'+':     cout << m << '+' << n << '='
                      << m + n << endl;
                break;
      case'/': cout << m << '/' << n << '='
                       << m/n << endl;
      case'*': cout << m << '*' << n << '='
                       << m * n << endl;
                break;
      case '%': cout << m << '%' << n << '='
                        << m% n << endl;
                break;
           case '-': cout << m << '-' << n << '='
                            << m-n << endl;
                break;
      default : cout << " not mathematical operation" << endl;
      }
      return 0;
}
```

والخرج هو :

```
enter two integers : 12 14↵
enter the operation : +↵
12 + 14 = 26
```

5- أعد كتابة البرنامج (3) باستخدام البنية switch

```
# include < iostream.h >
main()
{
      int mark;
      cout << " enter student mark:";
      cin >> mark;
      switch (mark / 10)
      {
      case 10: case 9: cout << 'A' << endl;
                        break;
      case 8 : cout << 'B' << endl; break;
      case 7 : cout << 'C' << endl; break;
      case 6 : cout << 'D' << endl; break;
      default : cout << 'F' << endl;
      }
      return 0;
}
```

والخرج هو :

```
enter student mark : 67↲
D
```

6- اكتب برنامجاً يختبر إذا كان الرمز المدخل هو أحد الحروف الصوتية أم أحد رموز المعاملات الحسابية ويطبع ما يلي:

أ – إذا كان الرمز المدخل صوتياً يطبع العبارة it is vowel

ب – إذا كان الرمز المدخل هو أحد رموز العمليات الحسابية يطبع العبارة "It is an operator"

ج – أي رمز آخر يطبع العبارة "It is something else"

ملاحظة : الأحرف الصوتية هي : a,e,i,o,u ورموز العمليات الحسابية هي : _,+,*,/,%

```
# include < iostream.h >
main()
{
    char str;
    cout << " enter a str :";
    cin >> str;
    switch (str)
    {
       case 'a' : case 'e' : case ''i' case '0' : case 'u' :
           cout << " It is a vowel " << endl;
           break;
       case '+' : case '-' : case '*' : case '/' : case '%' :
           cout << " It is an operator " << endl;
           break;
       default : cout << " It is something else " << endl;
    }
    return 0;
}
```

والخرج هو :

```
enter a str : u⮰
It is a vowel
```

7- اكتب برنامجاً يقرأ عمر المستخدم ويقوم بطباعة إحدى العبارات التالية :

أ – إذا كان عمر المستخدم أقل من 18 يطبع العبارة "you are a child".

ب – إذا كان عمر المستخدم أكبر من أو يساوي 18 وأقل من 56 يطبع العبارة "you are an adult".

ج – إذا كان عمر المستخدم أكبر من أو يساوي 65 وأقل من 120 يطبع العبارة : "you are a senior citizen"

د – فيما عدا ذلك يطبع العبارة "error age"

```
# include < iostream.h >
main()
{
    int age;
    cout << " enter your age:";
    cin >> age;
    if (age < 18)
    cout << " your are a child " << endl;
    else  if (age > = 18 && age < 65 )
     cout << "you are an adult " << endl;
        else  if (age > = 65 && age < 120 )
            cout << " you are a senior citizen " << endl;
              else
                cout << " error age " << endl;
    return 0;
}
```

وخرج البرنامج هو:

```
enter your age : 5⮰
your are a child
```

8- اكتب برنامجاً يدخل عددين صحيحين ويستعمل معامل التعبير الشرطي لاختبار وطباعة هل أحدهما من مضاعفات الآخر أم لا ويطبع العبارة "multiple" إذا تحقق الشرط وإلاّ يطبع العبارة "not multiple"

```
# include < iostream.h >
main()
{
     int N1 , N2;
     cout << " enter two integers not equal to zero:";
start : cin >> N1 >> N2;
        if ( N1 = = 0 || N2 = = 0 )
        goto start;
cout << ((( N1% N2 = = 0) || (N2%N1 = = 0))? "multiple": "not multiple")
     << endl;
        return 0;
}
```

والخرج هو :

```
enter two integers not equal to zero: 15  5 ↵
multiple
```

9- اكتب برنامجاً يوجد جذري معادلة من الدرجة الثانية $ax^2 + bx + c = 0$. يقبل البرنامج إدخال قيم حقيقية للمتغيرات الثلاثة c,b,a ويعين قيمة الجذرين حسب المعادلة التالية:

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

يجب التأكد من أن $a \neq 0$ وأن المقدار $b^2 - 4ac$ غير سالب.

```
// program to solve the equation a*x*x+b*x+c=0
        # include < iostream.h >
        # include < math.h > // needed for the sqrt() function
        main()
        {
```

```
float a,b,c,d,x,x1,x2,xre,xim;
cout << "enter coefficients of quadratic equation:";
cin >> a >> b >> c;
if (a = = 0)
{
    cout << " this is not quadratic equation : a = = 0" << endl
          << " It is first order equation" << endl;
    x =-c/b;
cout << "x =" << x << endl;
}
else
  {
    // second order equation
    d = pow (b,2)-4 * a * c;
    cout << "delta = " << d << endl;
    if (d > 0)
        {
            cout << " two Real Roots \n";
            x1 = (-b-sqrt (d))/2*a;
            x2 = (-b+sqrt (d))/2*a;
            cout << " x1 = " << x1 << endl;
            cout << " x2 = " << x2 << endl;
        }
    else if (d = = 0)
            {
                cout << " Double Root \n";
                x =-b/(2*a);
                cout << "x1 = x2 = " << x << endl;
            }
        else
            {
                cout << " Two complexs Roots \n";
                xre =-b/(2*a);
```

```
                    xim = sqrt (abs(d)) / (2*a);
                    cout << " x1 = " << xre << "+j" << xim<<endl;
                    cout << " x2 = " << xim << "-j" << xim<<endl;
                 }
         }
    return 0;
}
```

وخرج البرنامج هو :

```
enter coefficients of quadratic equation : 0  4  2 ↵
this is not quadratic equation : a = = 0
It is first order equation
x = - 0.5
```

10- اكتب برنامجاً يدخل درجة الحرارة مقدرة بالفهرنهايت. ويحدد هل هي مناسبة لجو الغرفة النظامي أم لا . فإذا كانت مناسبة يطبعها وبجانبها عبارة "good" وإذا كانت غير مناسبة يطبعها وبجانبها كلمة "Bad" . علما أن سلم درجات الحرارة بالفهرنهايت يتراوح ضمن المجال (212 , 32-) وأن درجة حرارة الغرفة النظامية هي ضمن المجال (25-15) درجة مئوية. قانون تحويل درجة الحرارة من حرارة فهرنهايت إلى مئوية هو:

$$C = (F - 32)\frac{5}{9}$$

حيث C درجة الحرارة المئوية و F درجة حرارة فهرنهايت.

```
// program to demonstrate the ideal environment temperature
# include < iostream.h >
main ()
{
    int  ctemp, ftemp;
    cout << " enter Fahrenheit please:";
    cin >> ftemp;
    if (ftemp < -32 || ftemp > 212)
    cout << "Temperature is out of range";
  else
    {
       ctemp = (ftemp-32 ) * 5/9;
           if (ctemp > = 15 && ctemp < = 25)
               cout << ctemp << " is good ";
           else
               cout << ctemp << "is Bad";
    }
  return 0;
}
```

وخرج البرنامج هو :

```
enter Fahrenheit please : 70↵
21  is good
```

11-اكتب برنامجاً يقوم بإدخال عدد صحيح بالترميز العشري ويحوله لمكافئه بالترميز الثنائي:

```
# include < iostream.h >
# include < math.h >
main( )
{
    int num10 , b , c;
    int num2 = 0 , i = 0;
    int num;
    cout << " enter a decimal number:";
    cint >> num10;
    num = num10;
start:  b = num10/2;
        c = num10%2;
        num2 = num2 + c * pow (10,i);
        i++;
        if (b = = 0)
        goto end;
        else
        {
            num10 = b;
            goto start;
        }
end:    cout << " the decimal number " << num
              << " is equal to " << num2 << " in binary " << endl;
    return 0;
}
```

والخرج هو :

```
enter a decimal number : 13↵
the decimal number 13 is equal to 1101 in binary
```

12- اكتب برنامجاً يسمح بإدخال إحداثيات نقطة (x,y). ويكون خرجه I وقيمتها صحيحة (1,2,3,4) وذلك حسب الربع الذي تقع فيه النقطة، وعلى النحو التالي:

تقع النقطة في الربع I

أو إحدى الرسائل التالية، وحسب موقع النقطة:

تقع النقطة على المحور x

تقع النقطة على المحور y

تقع النقطة في نقطة الأصل

```
# include < iostream.h >
main ( )
{
    int x , y , I = 0;
    cout << " enter x and y : ";
    cin >> x >> y;
    if (x > 0 && y > 0)
        I = 1;
    else if (x < 0 && y > 0)
            I = 2;
        else if (x < 0 && y < 0)
                I = 3;
            else if (x > 0 && y < 0)
                    I = 4;
                else if (x = = 0 && y ! = 0)
                    cout << " the point is on y " << endl;
                    else if (x! = 0 && y = = 0)
                        cout << " the point is on x " << endl;
                        else
                                cout << " the point is on the origin" << endl;
        if (I ! = 0)
            cout << " the point is on the " << I << " quarter"<<endl;
                << endl;
    return 0;
}
```

والخرج هو :

```
enter x and y : 0   5 ↵
the point is on y
```

13-اكتب برنامجاً يقوم بإدخال قيمة لمتغير صحيح يدعى B ثم يقوم بما يلي:

أ – إذا كانت قيمة B = 1 ، يسمح بإدخال قيمة x نصف قطر دائرة، ويحسب ويطبع مساحتها.

ب – إذا كانت قيمة B = 2 ، يسمح بإدخال قيمة الضلع x والارتفاع h ويحسب ويطبع مساحة المثلث المشكل منهما.

ج – إذا كانت قيمة B = 3 ، يسمح بإدخال طول ضلع المربع x ويحسب ويطبع مساحته.

د – يتم إنهاء البرنامج إذا كانت قيمة B لاتساوي أياً من القيم الثلاث السابقة.

```
# include < iostream.h >
main ( )
{
    int b;
    const float pi = 3.14;
    cout << " enter b value " << endl;
    cin >> b;
    if ( b = = 1)
    {
        float x;
        cout << " enter the radius" << endl;
        cin >> x;
        float circle = pi * x * x;
        cout << " circle area = " << circle << endl;
    }
else  if (b = = 2)
        {
            cout << " enter the triangle base and hight" << endl;
            float x,h;
            cin >> x >> h;
            float triangle = x*h/2;
            cout << " triangle area = "<< triangle << endl;
        }
else if (b = = 3)
        {
            cout << " enter the square side "<< endl;
            float x;
            cin >> x;
            cout << " square area = " << x*x << endl;
        }
    cout << " end of program " << endl;
    return 0;
}
```

14-اكتب برنامجاً يدخل ثلاثة أطوال صحيحة، ومن ثم يبحث في إمكانية تشكيل مثلث من هذه الأطوال. وإذا كان تشكيل المثلث ممكناً، فهل هو مثلث قائم الزاوية؟ خرج البرنامج إحدى الرسائل التالية :

1- لايمكن تشكيل مثلث	triangle can not be formed
2- يمكن تشكيل مثلث	triangle can be formed
3- يمكن تشكيل مثلث قائم الزاوية	right triangle can be formed

```
# include < iostream.h >
main ( )
{
    int a,b,c,t;
    cout << " enter three positive integer values:";
    cin >> a >> b >> c;
    // find the biggest value and assign it to a
    if ( a < b )
    {
      t = a; a = b ; b = t;
    }
    if ( a < c )
    }
    t = a; a = c; c = t;
    }
    if ( a > = (b+c))
      cout << " triangle can not be formed " << endl;
    else  if (a*a = = b*b+c*c)
           cout << " Right Triangle can be formed " << endl;
          else
           cout << " Triangle can be formed " << endl;
    return 0;
}
```

وخرج البرنامج هو :

```
enter three positive integer values : 5  4  3↵
Right Triangle can be formed
```

الفصل السادس

التكرار وحلقات التكرار

Repetition and Loops

توفر لغة C^{++} كسائر لغات البرمجة عدداً من الطرق تؤمن تكرار تنفيذ تعليمة أو أكثر من البرنامج عدداً من المرات وهي :

6-1- استخدام تعليمة الانتقال غير المشروط goto:

يمكن تكرار تنفيذ تعليمة أو مجموعة من التعليمات عدداً من المرات باستخدام ما يسمى عداد التحكم بالتكرار (Counter controlled repetition) وتعليمة الانتقال غير المشروط goto .

تأخذ تعليمة الانتقال غير المشروط الشكل التالي :

```
goto label;
```

حيث :

goto كلمة مفتاحية تعني الانتقال إلى.

label عنوان تعليمة سابقة أو لاحقة في البرنامج وهو اسم مقبول في اللغة ويكتب الاسم (label) متبوعاً بنقطتين :(colon) إلى يسار التعليمة المراد العودة أو الانتقال إليها والمتابعة بالتنفيذ بدءاً منها.

مثال (1) : اكتب برنامجاً يطبع الأعداد الصحيحة الخمسة الأولى باستخدام الانتقال غير المشروط.

```
# include < iostream.h>
main()
{
    int counter = 0;
loop: counter ++;
     cout << " counter : " << counter << " \n ";
     if(counter < 5)
          goto loop;
     cout << " complet counter :" << counter << " \n ";
     return 0;
}
```

شرح البرنامج :

بدأ البرنامج بتعليمة التصريح

```
int counter = 0;
```

حيث يستخدم كعداد المتحول الصحيح counter ويأخذ قيمة أولية مساوية للصفر يبدأ التكرار بالتعليمة :

```
loop: counter++;
```

تقوم هذه التعليمة بزيادة المتحول counter بالقيمة واحد ولها عنوان اسمه loop يلحق عنوان التعليمة بالرمز : (colon) . تتم طباعة المتحول بتعليمة الإظهار التالية :

```
cout << " counter : " << counter << "\n";
```

أما تعليمة الاختبار التالية :

```
if(counter < 5 )
```

فهي ضرورية لتحديد إما المتابعة بتكرار التعليمات المطلوب تكرارها أو المتابعة بتنفيذ تعليمات البرنامج. فإذا كانت قيمة counter أقل من 5 يعود لينفذ التعليمات بدءاً من التعليمة ذات العنوان loop وذلك تنفيذاً لتعليمة الانتقال غير المشروط goto loop; ، أما إذا كانت قيمة العداد counter تساوي أو أكبر من 5 عندها يتابع بتنفيذ التعليمة التالية في البرنامج وهي تعليمة الإظهار التالية:

```
cout << " complet counter : " << counter << " \n ";
```

بتنفيذ البرنامج السابق نحصل على الخرج التالي:

```
counter : 1
counter : 2
counter : 3
counter : 4
counter : 5
complet counter : 5
```

إن استعمال تعليمة goto يعطي برامج مشربكة وصعبة القراءة تعرف بكود المعكرونة المتداخلة (Spaghetti code) ولذا وجدت طرق أخرى سهلة القراءة والفهم وأكثر تطوراً مثل while , for.

2-6- البنية التكرارية while (while repetition structure) :

الشكل العام لبنية التكرار while

```
while ( condition )
  {
     statements;
  }
  next statement;
```

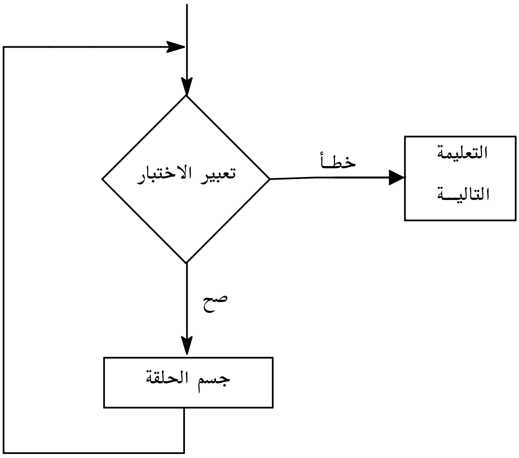
تبدأ البنية التكرارية بالكلمة المفتاحية (while) ثم تعبير الشرط أو الاختبار حيث يتم اختباره أولاً، لاحظ وجود القوسين حول الشرط المختبر. ومادام الشرط محققاً كرر تنفيذ التعليمات statements والتي تدعى جسم الحلقة وإلاّ تابع بتنفيذ التعليمة التالية next statement عندما يصبح الشرط غير محقق.

لاحظ أنه إذا تضمن جسم الحلقة تعليمة واحدة فقط، فإننا لانحتاج إلى الأقواس الكبيرة مثال:

```
while (n < 100 )
        n*=2;  //    one statement loop body, so no braces
```

ستستمر هذه الحلقة في مضاعفة قيمة المتغير n إلى أن لاتعود القيمة الناتجة أصغر من (أي عندما تصبح أكبر من أو تساوي) 100 ، عندها تتوقف الحلقة. إذا كانت n تساوي 1 عند بدء الحلقة، فستكون قيمتها عند الانتهاء 128 . حيث ستأخذ n القيم

في التسلسل التالي : 1 , 2 , 4 , 8 , 16 , 32 , 64 , 128 وعندها تتوقف الحلقة ، أي سينتقل التنفيذ إلى التعليمة التي تلي الحلقة، الشكل (6-1) التالي يبين مخطط سير العمليات لبنية while.



الشكل (6-1) **مخطط سير العمليات لبنية** while **التكرارية**

مثال (2): يمكن إعادة كتابة البرنامج في المثال (1) كما يلي:

```
# include < iostream.h>
main ()
{
    int counter = 0;
    while (counter < 5 )
       {
            counter ++;
            cout << " counter: " << counter << " \n ";
       }
   cout << " complet counter : " << counter << " \n ";
   return 0;
}
```

والخرج هو نفسه

```
counter : 1
counter : 2
counter : 3
counter : 4
counter : 5
complet counter : 5
```

مثال (3) : اكتب برنامجاً يطبع مربعات الأعداد الصحيحة والموجبة:

```
# include < iostream.h>
main ()
{
        int n ;
        cout <<"enter a positive integer number \n\t:";
start:  cin>>n;
        if(n<0)
          goto start;
          while (n > 0)
            {
                  cout << n << " square is " << n * n << endl<<"\t:";
                  cin >> n;
            }
     return 0;
}
```

خرج البرنامج

```
enter a positive integer number
            : 2↵
2 square is 4
            : 5↵
5 square is 25
            : 0↵
```

تم إدخال القيمة 2 الموافقة للشرط n > 0 . لذا سينفذ جسم الحلقة while أي يتم حساب مربع العدد 2 ، وقراءة قيمة جديدة لـ n ويعود التكرار حتى إدخال قيمة أقل أو تساوي الصفر فيصبح شرط الحلقة while غير محقق وينتهي تنفيذها.

3-6- البنية التكرارية do ... while (do – while repetition structure):

تشبه أسلوب التكرار للبنية while و for التي سندرسها لاحقاً إلاّ أنها تسمح بتنفيذ جسم البنية لمرة واحدة فقط حتى ولو لم يتحقق الشرط الذي يأتي في نهاية التنفيذ.

الشكل العام هو :

```
do
{
    statements;
}   while (condition);
next statement;
```

مثال (4): تواصل حلقة do... while في البرنامج التالي جمع عددين يدخلهما المستخدم، وتتوقف عن الجمع بإدخال القيمة صفر للعدد الأول.

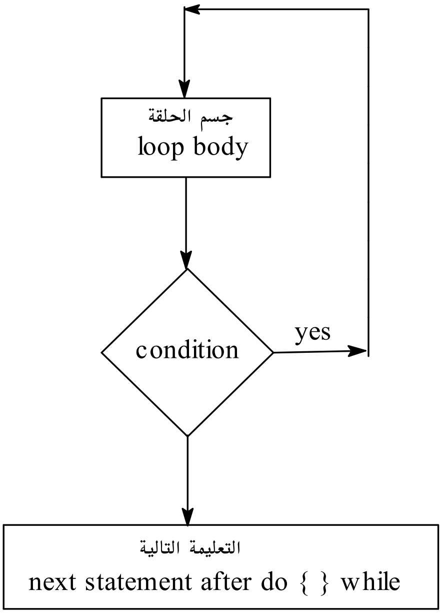
```
# include < iostream.h >
main()
{
    int x,y;
    do
    {
        cout << " \n enter two numbers (to quit, set first to 0):";
        cin >> x >> y;
        cout << "the sum is " << x + y ;
    } while ( x ! = 0 );
 return 0;
}
```

والخرج هو :

```
enter two numbers (to quit, set first to 0) : 22   13↵
the sum is 35
enter two numbers (to quit, set first to 0) : 13   0↵
the sum is 13
enter two numbers (to quit, set first to 0) 0   23↵
```

```
the sum is 23
```

الشكل (6-2) التالي يبين مخطط سير العمليات لبنية do ... while التكرارية.



الشكل (6-2) **مخطط سير العمليات لبنية** do…while **التكرارية**

مثال (5) : البرنامج التالي يدخل قيماً صحيحة أكبر من الواحد ويقوم بحساب مجموعها وطباعته. يتوقف عن الإدخال عند إدخال قيمة أقل من 1.

```
# include < iostream.h >
main ()
{ int n , sum = 0;
    cout << " enter integer numbers greater tham one:" << endl;
    do
    { cin >> n;
       sum + = n;
    } while (n > 1);
       cout << " the sum is = " << sum << endl;
    return 0;
}
```

والخرج هو:

```
enter integer numbers greater than one:
3  7  5  8  0  3  4↵
the sum is = 23
```

مثال (6): البرنامج التالي يحسب عاملي عدد صحيح موجب. حيث عاملي عدد صحيح n يعطى بالعلاقة:

$$n! = (n)\ (n-1)\ \ldots\ (3)\ (2)\ (1)$$

```
# include < iostream.h>
main ()
{ int n, m, nfact = 1;
     cout << "enter a positive integer value:";
     cin >> n;
     m = n;
     do
     { nfact * = n;
        n--;
     } while ( n > 1);
     cout << m << "factorial is = " << nfact << endl;
     return 0;
}
```

البرنامج يصرح عن متغيرين n و nfact ويعين قيمة أولية للمتغير nfact = 1 .
ثم يظهر عبارة تطلب إدخال قيمة صحيحة وموجبة n وبعد ذلك يقوم بضرب قيمة nfact بقيمة n وبقيم كل الأعداد الصحيحة والموجبة والتي هي أقل من n . لذلك عاملي العدد 5 هو :

$$5! = 5 * 4 * 3 * 2 * 1$$

والخرج في حالة n = 5 هو :

```
enter a positive integer value : 5↵
5 factorial is = 120
```

4-6- البنية التكرارية for

الشكل العام لهذه البنية هو :

```
 for (initial value; repetition condition; increment)
   {
         statements;
   }
next statement;
```

حيث:

initial value : القيمة الابتدائية لعداد الحلقة.

repetition condition : شرط استمرار تنفيذ الحلقة for وغالباً ما يحوي القيمة النهائية للعداد.

increment : الزيادة المنتظمة لعداد الحلقة (زيادة موجبة أو سالبة).

statements : تعليمة وحيدة أو مجموعة من التعليمات توضع بين قوسين كبيرين يتم تكرار تنفيذها عدداً من المرات يساوي عدد مرات تنفيذ الحلقة for.

تقوم بنية for بتنفيذ جسم الحلقة statements مبتدئة بقيمة للعداد تساوي قيمته الابتدائية. وفي المرة التالية، تزداد قيمة العداد بمقدار الزيادة ويتم اختبار قيمته الجديدة وإذا كانت محققة للشرط عندها يعاود التكرار. وينفذ جسم الحلقة، وهكذا حتى يصل عداد الحلقة لقيمة تحقق شرط الإنهاء والخروج من جسم الحلقة والمتابعة بتنفيذ التعليمة التالية next statement.

يمكن حساب عدد مرات تكرار تنفيذ الحلقة حسب القاعدة التالية :

$$\text{عدد المرات} = \left| \frac{\text{القيمة النهائية للعداد} - \text{القيمة الابتدائية للعداد}}{\text{الزيادة المنتظمة للعداد}} + 1 \right|$$

مثال (7):

لنعيد كتابة البرنامج السابق في المثال (6) لحساب عاملي عدد باستخدام بنية الحلقة for .

```
# include < iostream.h>
main ()
{
     int n, nfact = 1;
     cout << " enter a positive enteger value:";
     cin >> n;
     for (int i =2; i < =  n; i++)
          nfact * = i ;
     cout << n << " factorial is = " << nfact << endl;
     return 0;
}
```

تبدأ الحلقة بالكلمة المفتاحية for ثم قوسين (   ). التعليمة الأولى ضمن القوسين تحدد القيمة الأولية التي يأخذها عداد الحلقة وهي ;int i = 2. ثم التعليمة الثانية وهي شرط التكرار (i < = n). التعليمة الثالثة تحدد مقدار الزيادة وهي هنا واحد .i++

إذن طالما أن قيم i المتزايدة بواحد تحقق الشرط (i < = n) يتم تكرار تنفيذ جسم الحلقة وهو هنا تعليمة وحيدة ;nfact * = i . لذلك لاحاجة لوضع الأقواس { } . لاحظ ضرورة وجود الفاصلة المنقوطة بين التعليمات الثلاث ضمن القوسين ( ) للحلقة . for

إن الشرط الموجود داخل الحلقة (التعليمة الثانية هنا) والموجود بعد for يتم التحقق من صحته قبل تنفيذ جسم الحلقة. وهذا يضمن عدم تنفيذ جسم الحلقة إذا لم يكن شرطها محققاً . وبإدخال قيمة لـ n تساوي 5 يكون الخرج تماماً كما في المثال (6).

يمكن كما ذكرنا إنقاص عداد الحلقة بدل زيادته بأية قيمة نشاء مثال:

```
for ( j = 10; j > 0  ; j-- )
     cout << j << "  ";
```

البنية السابقة ستعرض القيم التالية:

```
10  9  8  7  6  5  4  3  2  1
```

تتمتع بنية for بمرونة عالية بشأن ما يمكن وضعه في التعابير الثلاثة مما يؤدي إلى توسيع إمكانية الاستفادة منها بطريقة أكفأ.

يمكن مثلاً استعمال عدة تعابير للتمهيد أو للاختبار. مثال:

```
for(j = 0 , total = 0  ;  j < 10  ;  j++ ,  total = total + j)
                    cout << total << "  ";
```

ستعرض هذه الحلقة ما يلي:

```
0   1   3   6   10   15   21   28   36   45
```

من أجل قيم j التالية : 0 ، 1 ، 2 ، ... 9 .

هنا الأجزاء الثلاثة لتعليمة for هي:

الجزء الأول التمهيدي : احتوى على تمهيد لعداد الحلقة j بالقيمة صفر وللمتغير total بالقيمة صفر أيضاً .

الجزء الثاني الشرط : احتوى على تعليمة شرطية واحدة وهي j < 10.

الجزء الثالث الزيادة : احتوى على تعليمة زيادة عداد الحلقة بواحد وعلى تعليمة زيادة المتغير total بالمقدار j .

لاحظ وجود الفاصلة العادية (,) بين تعليمات كل جزء من أجزاء تعليمة for

مثال (8) :

```
# include < iostream.h>
main ()
{
    int i,j;
    for (i=1, j=2; i * j<24;  i++)
       {
          j=i+1;
          j=j+2;
          cout << "\n" << "i=" << i << "\t j="<<j;
       }
```

```
   return 0;
}
```

والخرج هو :

```
    i=1        j=4
    i=2        J=5
    i=3        J=6
```

6-5- التكرار اللانهائي:

6-5-1- حلقة for اللانهاية

يمكن كتابة حلقة for التي تتكرر باستمرار ولاتتوقف إلاّ بوضع تعليمة توقفها مثل break التي سندرسها لاحقاً.

إن الشرط الذي ينظم عدد مرات التكرار في الحلقة تم حذفه من البرنامج التالي، ولذلك فإن الحلقة ستستمر بالتكرار إلى ما لانهاية في زيادة قيمة i وجمع الناتج مع قيمة sum .

```
# include < iostream.h>
main ()
{
        int i=1, sum=0;
        for (   ;    ;   )
        {
            sum + = i++;
            cout << "sum = " << sum << endl;
        }
   return 0;
}
```

البرنامج التالي يستمر بطباعة العبارة الموضحة إلى ما لانهاية أيضاً:

```
# include < iostream.h>
main ()
{
        for (   ;    ;   )
        cout <<" \n there is no god but Alah";
        return 0;
}
```

الشرط (if(sum = = 1000 في البرنامج التالي ليس له علاقة بالشرط المحذوف للحلقة وهو الذي يؤدي إلى توقف التكرار عندما يصل المجموع إلى 1000 ، وذلك باستخدام التعليمة break التي تنقل التنفيذ إلى خارج جملة for أي إلى تنفيذ تعليمة الطباعة cout خارج الحلقة for.

```
# include < iostream.h >
main  ()
{
  int i=1, sum = 0;
  for (  ;    ;   )
   {
     sum + = i++;
     if(sum = = 1000)
        break;
    }
  cout <<"sum = " << sum ;
  return 0;
}
```

ويكون الخرج هو :

```
sum = 1000
```

6-5-2- حلقة while اللانهائية:

يمكن أيضاً كتابة الحلقة while بحيث يكون الشرط المختبر صحيحاً دوماً وبالتالي تنفذ إلى ما لا نهاية.

والبرنامج التالي يستمر بالتنفيذ إلى مالانهاية حيث إن شرط الحلقة هو (1) وهو صحيح دوماً، ولولا وجود شرط آخر بعد الحلقة يطلب الخروج من التكرار إذا وصلت

قيمة counter لقيمة أكبر من 10 ، وذلك بتنفيذ التعليمة break لما توقف تكرار الحلقة while(1).

```
# include < iostream.h>
main ()
{
     int counter = 0;
     while (1)
     {
          counter ++;
          if (counter > 10)
               break;
     }
     cout << " counter : " << counter << "\n";
     return 0;
}
```

6-6- تعليمة break :

لقد مرت بنا تعليمة break في بنية الاختبار المتعدد وفي تعليمة الحلقة ولاحظنا أنها تنقل التنفيذ مباشرة إلى خارج بنية الاختبار أو التكرار.

مثال (9):

```
// using a break statement in a for structure
# include < iostream.h>
main ()
{
     int i;
     for (i=1; i < 100; i++)
     {
          cout << "i = "<< i << endl;
          if (i = = 10)
            break;          // break loop if only i=10
     }
```

```
    cout << " brok out of loop at i="<<i<<endl;
    return 0;
}
```

تعليمة break هنا تنفذ عند تحقق الشرط i=10 وتؤدي إلى توقف تكرار تنفيذ الحلقة والخروج منها والمتابعة بتنفيذ التعليمة التالية بعد الحلقة.

خرج البرنامج هو:

```
i = 1
i = 2
.
.
i = 9
i = 10
broke out of loop at i = 10
```

يتم عادة استخدام تعليمة break مع تعليمة if لتحديد حالة توقف التكرار ولو لم تستخدم تعليمة break مع تعليمة if في المثال السابق لتكرر تنفيذ الحلقة 99 مرة.

مثال (10):

```
# include < iostream.h>
main ()
{
  int i;
  for (i=1; i<=100; i++)
   {
     if (i%2= =0)
        break;       // break loop only if i is even
     cout << " i = " << i
            <<"does not cause the loop to break."<<endl;
   }
 cout << " i = " << i
       <<"cause the loop to break."<<endl;
 return 0;
}
```

خرج البرنامج هو :

```
i = 1 does not cause the loop to break
i = 2 cause the loop to break.
```

7-6-تعليمة الاستمرار continue :

تعمل تعليمة الاستمرار continue على تجاوز تنفيذ بقية تعليمات جسم الحلقة خلال الدورة الحالية والانتقال إلى الدورة الثانية للتنفيذ.

مثال(11):

```
// using the continue statement in a for structure
# include < iostream.h>
main ()
{
   int i;
   for ( i=1; i<10; i++)
     {
         if (i = = 4)
            continue; // skip remaining code in loop if only i=4
         cout << i <<"  ";
     }
  cout <<"\n used continue to skip printing the value of i=4";
  return 0;
}
```

ويكون الخرج:

```
1   2   3   5   6   7   8   9
used continue to skip printing the value of i=4
```

مثال (12) :

برنامج يقوم بطباعة جميع الأعداد الزوجية المحصورة بين 1 و 100 التي تقبل القسمة من دون باق على كل من 2 , 4 , 6.

```
// An example on continue statement
```

```
# include < iostream.h>
main ()
{
    int num;
    for(num=1; num < = 100; num++)
    {
        if(num%2)
          continue;           // will skip the rest of loop only if
                                //    num%2 not equal zero
        else if (num%4)
                continue;
             else if (num%6)
                    continue;
                  else
                    cout << num << "    ";
    }
  return 0;
}
```

الخرج هو :

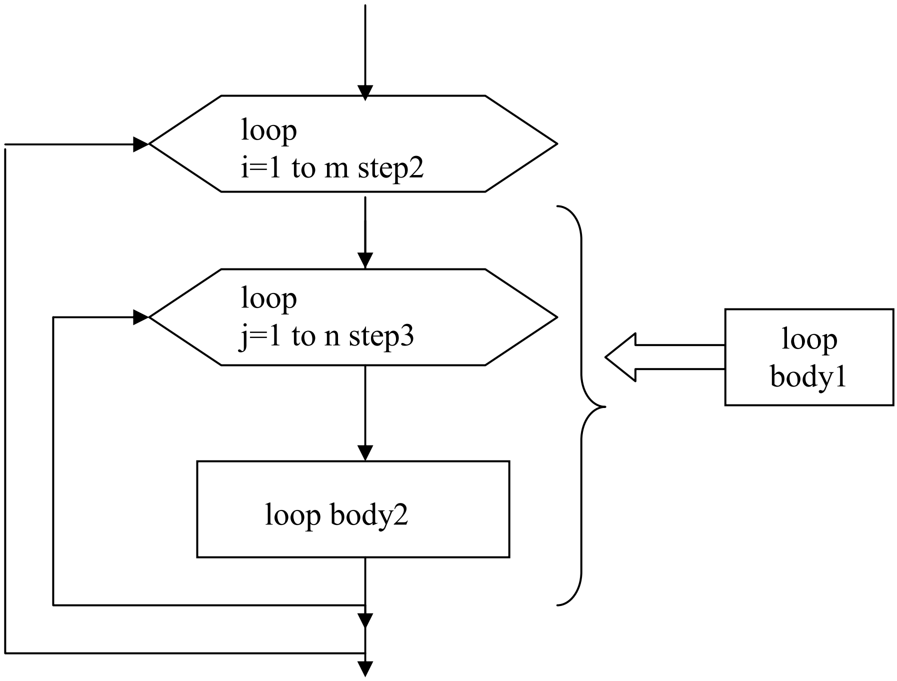
```
12  24   36   38   60   72   84   96
```

إذن الأمر break يجعل التنفيذ يتجاوز كل التعليمات المتبقية في جسم الحلقة التكرارية ويخرج من الحلقة إلى التعليمة التالية بعد الحلقة. والأمر continue يقوم بالشيء نفسه ماعدا أنه بدلاً من الخروج من جسم الحلقة فإنه يعاود مرة أخرى إلى بداية الحلقة ليبدأ تنفيذ التكرار التالي للحلقة.

6-8-الحلقات المتداخلة (Nested loops):

قد يحوي جسم الحلقة في بعض الأحيان على بعض التعليمات التي يتكرر تنفيذها عدداً من المرات في أثناء كل تنفيذ للحلقة التي تحتويها.

المخطط (3-6) التالي يبين أنه في أثناء كل تنفيذ لجسم الحلقة loop_body1 يتم تكرار مجموعة التعليمات loop_body2 عدداً من المرات مساوياً لـ n.



الشكل (3-6) مخطط سير العمليات للحلقات المتداخلة

6-8-1-حلقات for المتداخلة (nested for loops):

تأخذ صيغة حلقات for المتداخلة الشكل العام التالي:

```
for (                    )
        for (                    )
                for (                    )
                        statements;
```

وتكون في هذه الحالة التعليمة أو التعليمات جزءاً متكرراً مرتبطاً بالحلقة الداخلية والحلقة الداخلية تتكرر حسب أوامر الحلقة الخارجية التي تحتويها ... وهكذا.

مثال (13):

```
// An example of two nested loops
# include < iostream.h >
main ()
```

```
{
  int i,j;
  for (i=1; i<=3; i++)
   {
    for (j=1; j<=4; j++)
       cout << i*j<<"  ";
    cout << endl;
   }
   return 0;
}
```

ويكون الخرج :

```
1      2    3     4
2      4    6     8
3      6    9     12
```

لاحظ أن الحلقة الداخلية تنفذ 4 مرات لكل قيمة من قيم i في الحلقة الخارجية. التعليمة ;" "<<cout<<i * j يتكرر تنفيذها كالحلقة الداخلية 12 مرة وبعد كل تنفيذ للحلقة الداخلية بالكامل أي بعد أن تنفذ 4 مرات من أجل j=1,2,3,4 نفذ التعليمة ;cout << endl التي تؤدي إلى الانتقال إلى سطر جديد.

إذن عند الانتهاء من تنفيذ الحلقتين المتداخلتين تكون الحلقة الخارجية نفذت بعدد من المرات مساوٍ لعدد مرات تنفيذها، أما الحلقة الداخلية فتنفذ بعدد من المرات مساوٍ لعدد مرات تنفيذها مضروباً بعدد مرات تنفيذ الحلقة الخارجية. أي أن:

عدد مرات تكرار الحلقة الداخلية = عدد مرات تكرار الحلقة الداخلية × عدد مرات تكرار الحلقة الخارجية

2-8-6-حلقات while المتداخلة (Nested while loops):

تشبه حلقات while المتداخلة حلقات for المتداخلة.

مثال (14): لنُعِد كتابة المثال (13) باستخدام حلقات while المتداخلة

```
// example on nested while loops
# include <iostream.h>
main ()
  {
       int i,j;
       i=j=1;
          while(i<=3)
          {
             while (j<=4)
             {
                   cout << i*j<<"  ";
                   j++;
             }
             cout  << endl;
             i++;
             j=1;
          }
     return 0;
   }
```

الحلقة الخارجية

الحلقة الداخلية

ويكون الخرج تماماً كما في المثال (13):

```
1   2   3   4
2   4   6   8
3   6   9   12
```

الحلقة الخارجية i تنفذ ثلاث مرات ، بينما الحلقة الداخلية j تنفذ 4 مرات في كل مرة تنفذ فيها الحلقة الخارجية، أي تنفذ 12 مرة.

يمكن أن تتداخل حلقة while مع حلقة for كما في المثال التالي :

مثال (15) : لنُعِد كتابة المثال (13) باستخدام تداخل حلقة while مع حلقة for.

```
// An example on while – for nested loops
# include < iostream.h>
main ()
{
  int i,j;
  i=1;
  while (i < = 3)
   {
     for (j=1; j < = 4; j++)
         cout << i*j<< "   ";
     cout << endl;
     i++;
    }
  return 0;
}
```

وهو مطابق لمثال حلقتي while المتداخلة ويعطي نفس الخرج.

```
1       2     3      4
2       4     6      8
3       6     9      12
```

9-6- تمارين :

1- ماذا يفعل البرنامج التالي وما هو خرجه :

```
# include < iostream.h >
main()
{
     int x,y;
     cout << " enter two integer in the range 1_10:";
     cin >> x >> y;
     for (int i=1; i<=y; i++)
     {
        for (int j=1; j<=x; j++)
            cout << '@';
        cout << endl;
     }
     return 0;
}
```

يطلب البرنامج إدخال قيمتين صحيحتين y وx ضمن المجال (10-1) ثم يطلب طباعة الرمز '@' من أجل تغيرات i من (y-1) وتغيرات j من (x-1) . وهذا يؤدي إلى طباعة الرمز '@' × مرة ويكرر ذلك في y سطر.

أي أن الخرج هو :

```
enter two integer in the range 1_10 : 4  3↵
@ @ @ @
@ @ @ @
@ @ @ @
```

2- اكتب برنامجاً يحسب مجموع مربعات الأعداد الموجبة والصحيحة والأقل أو تساوي العدد المدخل num.

```
# include < iostream.h >
main( )
{
     int num, sum = 0;
     cout << " enter a poisitive integer number:";
start: cin >> num;
     if (num < = 0)
     go to start;
     for (int i=1; i<=num; i++)
        sum += i*i;
     cout << " the sum of first " << num << " squares is:"
            << sum << endl;
     return 0;
}
```

والخرج هو :

```
enter a positive integer number : 5↵
the sum of first 5 squares is: 55
```

4- اكتب برنامجاً يكرر إدخال عدد صحيح، موجب ويطبعه ويتوقف عن الإدخال إذا كان العدد المدخل من مضاعفات العدد (3) ثم يطبع العبارة outside of loop

```
#include<iostream.h>
```

```
main()
{
   int num;
   for (  ;  ;  )
    {
        cout << " enter a positive integer number:";
start:  cin >> num;
        if (num <= 0)
        goto start;
        else if (num%2 == 0)
              {
               cout << " the even positive integer is:" << num << endl;
               if (num%3 == 0)
               break;
              }
             Else
              cout << " bottom of loop \n" << endl;
   }
   cout << " outside of loop " << endl;
   return 0;
}
```

والخرج هو :

```
enter a positive integer number:11↵
bottom of loop
enter a positive integer number:8↵
the even positive integer is: 8
enter a positive integer number: 12↵
the even positive integer is: 12
outside of loop
```

4- اكتب برنامجاً يسمح بإدخال أعداد صحيحة موجبة، ويتوقف عن الإدخال بإدخال عدد سالب، ثم يوجد القيمة العظمى والقيمة الصغرى منها مع ترتيب إدخالهما، ويحسب المتوسط الحسابي للأعداد المدخلة.

حيث num اسم المتغير المدخل ، count عدد القيم المدخلة ، max القيمة العظمى، min القيمة الصغرى، imax ترتيب القيمة العظمى ، imin ترتيب القيمة الصغرى ، sum مجموع القيم و ave المتوسط.

```
# include < iostream.h >
main( )
{
     int max, min, sum = 0, num;
     int imax, imin, count = 1;
     float ave;
     cout << " enter different positive number and negative value to end:"
           << endl;
     cout << count << " : ";
start: cin >> num;
     if(num < 0)   goto start;
     min = max = num;
     imin = imax = 1;
     do
     {
        sum = sum + num;
        count++;
        cout << count << ":";
        cin >> num;
        if (num < 0)
          break;
        if (num > max)
            {
                max = num;
                imax = count;
             }
         if (num < min)
           {
             min = num;
             imin = count;
           }
      } while (num > = 0);
      -- count;
      cout << " sum = " << sum << " \t count = "
            << count << endl;
      ave = float (sum)/count;
      cout << " max = " << max << " \t imax = " << imax << endl;
      cout << "min = " << min << " \t imin = " << imin << endl;
      cout << "ave =" << ave << endl;
      return 0;
}
```

والخرج هو :

```
enter different positive number and 0 to end;
1 : 3↵
2 : 6↵
3 : 10↵
4 : 12↵
5 : 20↵
6 : 4↵
7 : 9↵
8 : -1↵
sum = 64       count = 7
max = 20       imax = 5
min = 3         imin = 1
ave = 9.14286
```

5- اكتب برنامجاً لإيجاد الجذر التربيعي الصحيح لأي عدد معطى. الجذر التربيعي هو العدد الصحيح الذي يكون تربيعه أقل من أو يساوي العدد المعطى.

نبحث في طريقة الحل المتبعة عن أكبر عدد تربيعه أقل أو يساوي العدد المعطى، عند ذلك يكون هو الجذر التربيعي للعدد المعطى.

```
# include < iostream.h >
main( )
{
     float x;
     cout << "enter a positive number:";
     cin >> x;
     for (int i=1; i*i<=x, i++)
        {
            if (i*i<=x)
              continue;
            else
              break;
        }
        cout << "the integer square root of " << x
              << "is" << i-1 << endl;
     return 0;
}
```

وخرج البرنامج هو :

```
enter a positive number: 630.15↵
the integer square root of 630.15 is 25
```

نبدأ بـ i = 1 وتستمر في زيادة i حتى يكون مربع i أكبر من x ، عندما تنتهي الحلقة التكرارية for تكون i هي أصغر عدد صحيح مربعه أكبر من العدد المدخل x لذلك فإن n-1 هو الجذر التربيعي الصحيح لـ x .

لاحظ أن جسم الحلقة كان بالإمكان حذفه مع المحافظة على وضع الفاصلة المنقوطة (;) فقط بالشكل التالي:

```
for (int i = 1; i * i< = x; i++)
         ;              // the null statement
```

6- اكتب برنامجاً يُدخل عدداً صحيحاً موجباً ويطبعه كما أُدْخلَ ، ثم يعكس ترتيب أرقامه ويطبعه ثانية بعد إجراء عملية العكس

```
# include < iostream.h >
main()
{
    long numi, d, numo = 0, m;
    cout << " enter a positive integer number:";
    cin >> numi;
    m = numi;
    while (numi > 0)
    {
       d = numi%10;      // d will be the right_most digit of numi
       numi /=10;        // then remove that digit from numi
       numo = 10*numo +d;      // and append that digit to numo
    }
    cout << " The reverse number of " << m << " is = "
          << numo << endl;
    return 0;
}
```

والخرج هو :

```
enter a positive integer number: 12345↵
The reverse number of 12345 is = 54321
```

7- نص البرنامج نفسه السابق ولكن مع طلب تحديد عدد أرقام العدد المدخل:

```
# include < iostream.h >
# include < math.h >
main()
```

```
{
    int d, k = 0;
    long numi, m, numo = 0;
    cout << " enter a positive integer number:";
    cint >> numi;
    m = abs(numi);
start: m/=10;
    K+=1;          // k is to count
    if (m > 0)     // the number of digit in numi
    goto start;
    cout << " the number of digit in numi is = " << k << endl;
    for (int i=1; i< =k; i++)
    {
       d = numi%10;
       numo = numo +d * pow(10, (k-i));
       numi /=10;
    }
    cout << " the reverse number is: " << numo << endl,
    return 0;
}
```

والخرج هو :

```
enter a positive integer number: 123456⏎
the number of digit in numi is = 6
the reverse number is : 654321
```

8- العدد المتماثل هو العدد الذي يعطي القيمة نفسها بقراءته من الطرفين. مثال 1123211 و 1234321 اكتب برنامجاً يقبل إدخال عدد صحيح غير محدد عدد خاناته ويختبر هل هو متماثل أم لا . يتوقف عن الإدخال بإعطاء القيمة صفر للعدد المدخل. يطبع البرنامج العبارة المناسبة حسب نوع العدد متماثل أم لا .

```
# include < iostream.h >
# include < math.h >
```

```
main()
{
 int d,k,l;
 long m,n,numi,numo=0;
 while (1)
 {
   cout << " enter a positive integer number :";
   cin >> numi;
   if (numi = = 0)
     break;
   cout << " the positive integer number is:"<< numi << endl;
   m = n = numi;
   k = l = 0;
start:  m/=10;
  k+=1;         // k is to count the
 if (m>0)      // number of digit in numi
   goto start;
 cout << " k = " << k << endl;
 for (int i=1, i<=k; i++)
 {
     d = numi % 10;
     numo=numo+d*pow(10,(k-l));
       numi = numi / 10;
 }
 if (numo = = n)
     cout << " the number " << num0 << " is palindrome"<<endl;
else
     cout << " the number " << n
            << " is not palindrome"<<endl;
}
return 0;
}
```

والخرج هو :

```
enter a positive integer number : 12321↵
the positive integer number is : 12321
k = 5
the number 12321 is palindrome
enter a positive integer number : 1234↵
the positive integer number is : 1234↵
k = 4
the number 1234 is not palindrome
enter a positive integer number : 0↵
```

9- اكتب برنامجاً لإيجاد القاسم المشترك الأعظم لعددين صحيحين موجبين.

تعتمد خوارزمية الحل (خوارزمية الإقليدي) على تحويل رقمين صحيحين (m,n) إلى رقمين (d,0) بالقسمة المتتالية للعدد الكبير m على العدد الصغير n مع استبدال الأكبر بباقي القسمة (d) . عندما يصبح الباقي صفراً فإن الرقم الآخر يكون هو القاسم المشترك الأعظم للعددين الأصليين (وكذلك للأزواج المتوسطة من الأرقام). الخوارزمية تتطلب أن يكون العدد الأول أكبر من الثاني وإلاّ وجب استبدالهما وهذا ما تم تضمينه في البرنامج.

```
# include < iostream.h >
main ()
{
     int m,n,r;
     char ch;
     while(1)
     {
        cout << " enter two positive integers :";
        cin >> m >> n;
        if (m<n)
        {
            int temp = m;       // make m > = n
                 m = n;
                 n = temp;
        }
        cout << " the g.c.d. of " << m << " and " << n << " is ";
        while ( n > 0 )
         {
            r = m% n;
            m = n;
            n = r;
         }
        cout << m << endl;
        cout << " enter y to continue or n to finish : ";
        cin >> ch;
        if (ch = = 'y')
            continue;
        else
            break;
    }
    return 0;
}
```

والخرج هو :

```
enter two positive integers : 300 72↵
the g.c.d of 300 and 72 is 12
enter y to continue or n to finish : n↵
```

10- اكتب برنامجاً يحسب ويخرج الفرق بين القيمة التي تعطى بوساطة التابع العلاقـاتي sin( ) والقيمة التي تعطى باستخدام الحدود العشرة الأولى للمتوالية التالية لجيب الزاوية x (بالراديان) المدخلة إلى البرنامج بالدرجات.

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!} - \dots$$

اجعل البرنامج يقوم بحساب الزوايا الموجبة وذات القيمة الأقـل مـن $\frac{\pi}{2}$ راديـان فقط (1 درجة = $\pi/180$).

```
# include < iostream.h >
# include < math.h >
main ( )
{
     const float pi=3.14;
     float xd , yr , l , dif , sum = 0;
     int i, j , m , g = 1;
     long int f;
     cout << " enter the angle in degrees and less than π/2:";
start: cin >> xd;
     yr = xd * pi/180;
     cout << " yr = " << yr << endl;
     if (yr < 0 || yr > pi/2)
        goto start;
     l = sin (yr);
     cout << " l= " << l<< endl;
     for (i = 1; i < = 10; i++)
     {
        m = 2 * i -1;
        f = 1;
        for (j = 1; j < = m; j++)
            f = f * J;
        sum = sum + g * pow (yr , m)/f;
        g = -g;
     }
cout << " sum = "<< sum << " ℓ= " << ℓ << endl;
     dif = fabs (sum – l);
     cout << " dif = " << dif << endl;
     return 0;
}
```

والخرج هو :

```
enter the angle in degrees and less than π/2 : 87.5↵
yr = 12.52639
l = 0.999014
```

```
sum = 0.998981    ℓ = 0.999014
dif = 3.29018e-05
```

11- الأعداد الأولية هي الأعداد التي لاتقبل القسمة من دون باقي إلاّ على نفسها وعلى الواحد. يمكن اختبار أي عدد إن كان أولياً أم لا وذلك باختبار الباقي عن ناتج قسمته على كافة الأعداد من 2 ولغاية جذره. وإن كان أي عدد باقي يساوي الصفر فالعدد لايكون أولياً.

اكتب برناجاً يكون خرجه كافة الأعداد الأولية من 1 ولغاية 100.

```
# include < iostream.h >
# include < math.h >
main ( )
{
    for(int i = 1, i < = 100; i++)
    {
       int k = 0;
       for (int j = 2; j < = sqrt (i); j++)
          {
              if (i% j = = 0)
                  k = 1;
          }
          if ( k = = 0)
          cout << i << "\t";
       }
    cout << endl;
    return 0;
}
```

وخرج البرنامج هو الأعداد الأولية التالية :

```
1,2,3,5,7,11,13,17,19,23,29,31,37,41,43,47,53,59,61,67,73,79,83,89,97
```

12- يملك العدد (3025) الخاصية التالية : مربع مجموع العددين المكونين من الرقمين في اليمين والرقمين في اليسار يساوي العدد نفسه أي :

$$3025 = (30 + 25)^2 = (55)^2 = 3025$$

اكتب برنامجاً يوجد جميع الأعداد المكونة من أربعة أرقام، والتي تحقق هذه الخاصية، ويطبعها:

```
# include < iostream.h >
# include < math.h >
main ( )
{
    int i, n, m;
    for (i = 1000; i < 10000; i++)
    {
      n = i/100;
      m = i %100;
      if (i = = pow (n+m,2))
          cout << i << "\t";
    }
      cout << endl;
    return 0;
}
```

والخرج هو :

```
2025        3025        9801
```

13- هنالك طرق عديدة لتوليد مصفوفة من الأعداد العشوائية . منها الطريقة التالية:

أ – نختار قيمة ابتدائية صحيحة غير زوجية ولاتنتهي بالرقم خمسة.

ب – نختار مضروباً ثابتاً.

ت – من حاصل جداء القيمة الابتدائية والمضروب، نأخذ الأرقام الأربعة الأقل مرتبة لتشكيل أول عدد عشوائي.

ث – يتم إعادة ضرب العدد العشوائي الجديد بالمضروب الثابت. وتؤخذ الأرقام الأربعة الأقل مرتبة من حاصل عملية الجداء الجديدة لتكون ثاني عدد عشوائي بالمصفوفة.

ج – تكرر العملية ث حسب عدد الأعداد العشوائية المطلوب :

اكتب برنامجاً لتوليد وطباعة 100 عدد عشوائي وفق الطريقة السابقة:

```
# include < iostream.h >
main( )
{
      long int n,b,d;
      const int m = 5;
start:cin >> n;
      if (n%2 ! = 0 && n% 5 ! = 0)
        {
           for (int i = 1, i< = 100; i++)
                {
                     b = n * m;
                     d = b%10000;
                     cout << " the random no " << i << " is "
                           << d << endl;
                     n = d;
                }
        }
     else
        goto start;
     return 0;
}
```

والخرج هو مئة عدد عشوائي أولهم هو العدد المدخل غير الزوجي ولاينتهي بالرقم خمسة مضروباً بالثابت 5. ويستمر تكرار ضرب الناتج بالثابت وأخذ العدد العشوائي المكون من الأرقام الأربعة الأقل مرتبة من الناتج حتى الحصول على مئة عدد عشوائي.

14- اكتب برنامجاً يدخل عدداً صحيحاً مكوناً من خمسة أرقام ويطبعه ويطبع أرقام العدد المدخل كل على حدة : مثلاً إذا كان الدخل هو العدد 12345 يكون الخرج :

12345

```
1 , 2 , 3 , 4 , 5 ,
# include < iostream.h >
# include < math.h >
main()
{
     int b , i , a;
     long int num;
     cout << " enter a five digit integer number:";
start: cin >> num;
     if (num > 99999 || num < 10000)
     goto start;
     for (i = 4; i < = 0; i--)
        {
            a = pow(10,i);
            b = num/a;
            cout << b << " , ";
            num% = a;
        }
        cout << endl;
     return 0;
}
```

والخرج هو :

```
enter a five digit integer number: 32415⏎
3 , 2 , 4 , 1 , 5
```

الفصل السابع

المصفوفات والسلاسل

Arrays & Strings

7-1- مقدمة :

المصفوفة هي الطريقة التي تمكن المبرمج من الدلالة على مواقع أو خلايا تخزين متتالية باستخدام اسم متغير وحيد. وللوصول إلى القيمة المخزنة في موقع محدد من المصفوفة وهو ما يسمى بعنصر المصفوفة لابدّ من وجود معلومة إضافية تدعى الدليل index أو subscript تدل على ترتيب ذلك العنصر ضمن المصفوفة.

درسنا حتى الآن كيفية التعامل مع متغيرات وحيدة القيمة وسنتعرض في هذا الفصل إلى المصفوفات التي تتعامل مع اسم متغير وحيد ولكنه يحوي عدة متغيرات (عناصر) من النوع نفسه. ولتعريف المترجم أن هذا الاسم هو لمصفوفة أي لمجموعة من القيم من النوع نفسه علينا أولاً إبلاغه عن نوع هذه ا لقيم والاسم الذي سنطلقه عليها وهو اسم مقبول بلغة ++C وكذلك عدد العناصر المشكلة لها أي حجمها.

من قبل لو كان لدينا خمسة قيم صحيحة ونريد إدخالها للحاسب ومن ثم طباعتها بعكس ترتيب إدخالها لكان علينا كتابة البرنامج التالي الذي يصرح عن خمسة متغيرات مختلفة القيم ومن النوع نفسه.

```
# include < iostream.h >
main ()
{
    int first, second, third, fourth, fifth;
    cin >> first >> second >> third
        >> fourth >> fifth;
    cout << fifth << fourth << third
```

```
        << second << first << " \ n ";
    return 0;
}
```

تصور لو أننا أردنا التعامل مع 50 قيمة بدلاً من 5 فقط. واضح أن الأمر سيكون متعباً من حيث إعطاء أسماء مختلفة لتلك القيم ومن حيث الزمن اللازم لإدخال البرنامج وتنفيذه.

7-2- التصريح عن المصفوفة:

يتم التصريح بلغة ++C عن المصفوفة تماماً كالتصريح عن المتغيرات. وتأخذ تعليمة التصريح الشكل العام التالي:

```
Data_type  array_name [number_of_elements];
```

حيث :

Data_type : هو أحد أنواع المتغيرات التي درسناها. لايسمح بالنوع void.

array_name : اسم مقبول في لغة ++C.

number-of_elements : هو حجم المصفوفة أي العدد الأعظمي لعناصرها. ويأخذ قيمة صحيحة أو اسماً لثابت صحيح.

يمكن التصريح عن أكثر من مصفوفة في تعليمة تصريح واحدة. التعليمة التالية:

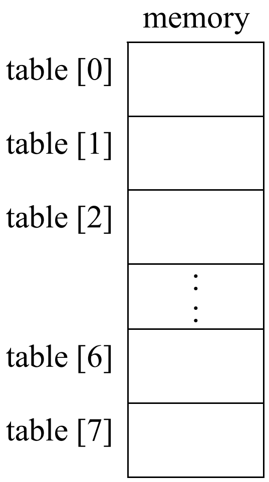
```
int b[10], x[2];
```

تصرح عن مصفوفتين الأولى b وعدد عناصرها 10 تأخذ الأسماء ,[b[0 b[1],...,b[9] والثانية x وعدد عناصرها 2 العنصر الأول x[0] والثاني x[1].

المثال التالي يصرح عن مصفوفة من نوع صحيح واسمها table وذات 8 ثمانية عناصر.

```
int table [8];
```

وتشغل هذه المصفوفة مواقع متتالية من التخزين في الذاكرة، يمكن تمثيلها بالشكل التالي:



يمكن الرجوع إلى كل عنصر من عناصر المصفوفة بذكر اسم المصفوفة ووضع دليل العنصر المطلوب الرجوع إليه بين قوسين مربعين. العنصر الأول في المصفوفة يأخذ الدليل 0 (صفر) دوماً. والعنصر الأخير يأخذ دليلاً يساوي عدد عناصر المصفوفة مطروحاً منه واحد. في مثالنا السابق العنصر الأخير هو table[7] حيث : 8 – 1 = 7 .

يمكن التعامل مع اسم عنصر المصفوفة تماماً كالتعامل مع أسماء المتغيرات وحيدة القيمة من النوع نفسه.

الأمثلة التالية لتعليمات صحيحة تتضمن التعامل مع أسماء متغيرات ذات دليل. أي مع أسماء لعناصر مصفوفة.

```
Sum2 = table[0] + table [2];
```

تعليمة تؤدي إلى جمع قيمتي العنصرين الأول والثالث للمصفوفة table وخزن الناتج في المتغير sum2 .

```
table [7] = 7;
```

تعين القيمة 7 للعنصر الأخير من المصفوفة المصرح عنها table

```
if (table[0]> table[1])
cout << " first is greater than second \ n";
```

تختبر التعليمة الأولى إذا كان العنصر الأول للمصفوفة table أكبر من العنصر الثاني وإذا كان محققاً تتم طباعة العبارة "first is greater than second" نتيجة تنفيذ التعليمة الثانية

3-7- المصفوفة أحادية البعد (one dimensional array):

المصفوفة ذات البعد الواحد a[n] تمثل جبرياً بشكل أفقي:

$$[\,a_1 \quad a_2 \quad a_3 \quad \ldots \quad a_n\,]$$

أو بشكل عمودي :

$$\begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_n \end{bmatrix}$$

أما في لغة C++ فتأخذ الشكل التالي:

Data_type array_name [size];

ويكون دليل العنصر الأول كما ذكرنا سابقاً صفر دوماً. ويحجز للمصفوفة بايتات متتالية من الذاكرة. يمكن حساب عدد البايتات المطلوبة لتخزين عناصر مصفوفة ما من العلاقة التالية:

no_of_storage_byte=no_of_array-elements*size_of-array_elements

حيث:

no_of_storage_byte : عدد المواضع اللازمة لتخزين عناصر المصفوفة بالبايت.

no_of_array_elements : عدد عناصر المصفوفة.

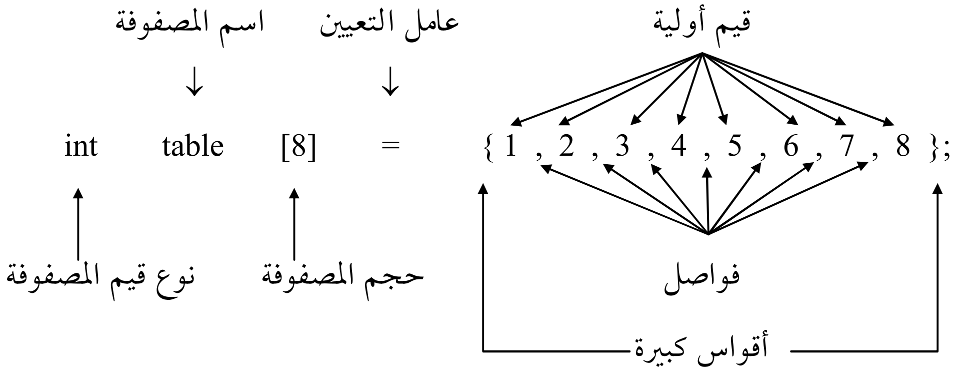
size_of_array_element : عدد المواضع اللازمة لتخزين عنصر واحد من المصفوفة بالبايت.

التعليمة int table[8] تحتاج إلى 16 بايت حيث إن العناصر من نوع int وتحتاج 2 بايت لتخزين كل عنصر ولدينا ثمانية عناصر في المصفوفة table .

1-3-7- إعطاء قيم ابتدائية لعناصر مصفوفة أحادية (Array initialization):

يمكن إعطاء قيم ابتدائية لعناصر المصفوفة في أثناء التصريح عنها وذلك بأن نتبع تعليمة التصريح بإشارة (=) وبقائمة من القيم يفصل بين عناصرها الموجودة ضمن قوسين كبيرين { } فواصل.

مثال:



تعبر قيم القائمة السابقة عن القيم الابتدائية Initializers لكل عنصر من عناصر المصفوفة.

ملاحظات:

1) إذا أعطينا قيماً ابتدائية أقل من عدد عناصر المصفوفة فإنه يتم إعطاء قيم ابتدائية مساوية للصفر لبقية العناصر مثال:

int table [8] = { 1,2,3,4,5};

العناصر الثلاثة الأخيرة من المصفوفة table ذات الدليل 7,6,5 تأخذ القيمة صفر.

table[5] = table [6] = table [7] = 0

حيث قيم العناصر الخمسة من المصفوفة table معطاة بشكل صريح ضمن القائمة المحصورة ما بين القوسين الكبيرين بعد إشارة (=) في تعليمة التصريح.

int n[10] = { 0 }; // initialize 10 integers to 0.

يقوم التصريح السابق ظاهرياً بإعطاء القيمة 0 إلى أول عنصر من عناصر المصفوفة n وضمنياً يعطي القيمة صفر أيضاً لبقية عناصر المصفوفة.

2) التصريح التالي :

```
int n[ ] = { 1,2,3,4,5 };
```

لم يحدد عدد عناصر المصفوفة. أي أنه تم حذف حجمها. لاحظ وجود الأقواس المربعة [ ] بدون قيمة داخلها. وبالتالي يصبح عدد عناصر هذه المصفوفة مساوياً لعدد القيم الابتدائية المعطاة ضمن القائمة الملحقة بالتصريح وهي هنا خمس قيم.

3) من الخطأ إعطاء قيم ابتدائية تزيد في عددها على عدد عناصر المصفوفة. لأن المترجم سيعترض على ذلك، التصريح التالي :

```
int n[5] = { 1,2,3,4,5,6 };
```

يعين ست قيم ابتدائية لمصفوفة n مصرح عنها ذات 5 عناصر فقط . وهذا يسبب خطأً قواعدياً في اللغة ويدعى (syntax error).

4) يسمح باستخدام اسم لقيمة ثابتة فقط في التصريح عن حجم المصفوفة وهذا يجعل البرنامج أكثر قابلية لتغيير الحجوم عند الحاجة.

```
const int array_size = 10;
int n[array_size] = { 1,2,3,4,5,6,7,8,9,10};
```

تعليمة التصريح الأولى تصرح عن اسم ثابت صحيح مساوٍ لـ 10 . ثم استخدم هذا الاسم array_size ليعبر عن حجم المصفوفة n في تعليمة التصريح الثانية.

7-3-2- إدخال وإخراج قيم عناصر المصفوفة أحادية البعد:

(Input & output the array elements):

يمكن إدخال قيم عناصر المصفوفة وتخزينها في مواقع متتالية في ذاكرة الحاسب باستخدام تعليمة الإدخال << cin مع أي من بنى التكرار المدروسة سابقاً. بحيث

يستخدم متغير حلقة التكرار كدليل لعنصر المصفوفة . كذلك الأمر بالنسبة لعملية الإخراج أو الطباعة حيث يستخدم أمر الطباعة >> cout مع أي من بنى التكرار.

مثال (1) : البرنامج التالي يدخل أربع قيم للمصفوفة year الصحيحة. ثم يطبعها بعكس ترتيب إدخالها.

```
# include < iostream.h >
main ()
{
    const int size = 4;
    int year [ size ];
    cout << " enter 4 values for array year :\ n";
    for (int i = 0;  i < size; i++)
        cin >> year [i];
    cout << " print the 4 entered values in reverse order :\ n";
    for (i = size – 1; i > = 0; i --)
        cout << " year [ " << i << " ] = "  << year [i] << endl;
  return 0;
}
```

بتنفيذ البرنامج السابق وإدخال القيم التالية : 1999 ,2000 ,2001 ,2002

يكون الخرج هو القيم المدخلة نفسها ولكن بترتيب عكسي وذلك كما يلي :

```
enter 4 values for array year:
1999   2000   2001   2002⏎
print the 4 entered values in reverse order:
year [3] = 2002
year [2] = 2001
year [1] = 2000
year [0] = 1999
```

مثال (2) : البرنامج التالي يدخل أربع قيم صحيحة ويعينها للمصفوفة age ثم يطبعها:

```
# include <iostream.h>
main()
{
    int age[4];
    for (int i = 0; i<4; i++)
       {
          cin >> age [i];
          cout << "age ["<< i <<" ] =" << age[i] << " \t";
       }
    return 0;
}
```

لو أدخلت القيم التالية 10 , 11 , 12 , 13 في سطر واحد لكان الخرج كما يلي:

```
10   11   12   13↵
age[0]=10             age[1]=11             age[2]=12             age[3]=13
```

أما لو أدخلت كل قيمة في سطر منفصل لكان الخرج كما يلي:

```
10↵
age[0]=10                11↵
age[1]=11                12↵
age[2]=12                13↵
age[3]=13
```

تذكر أنه يمكنك تعيين قيم لعناصر المصفوفة بتعليمة التصريح، ولكن المترجم لايقبل إعطاء عدد من القيم أكثر من عدد عناصر المصفوفة المحدد بالتعليمة نفسها. لكن ماذا لو أدخلت قيماً لعناصر المصفوفة عن طريق تعليمة الإدخال <<cin؟

إن المترجم ينفذ إدخال القيم وتخزينها حتى لو تجاوزت عدد عناصر المصفوفة.

لو كان لدينا المصفوفة الحقيقية A[4] وكتبنا التعليمات التالية:

```
float A[4];
for (int j = 0; j< = 4; j++)
      cin >> A[j];
```

ينفذ المترجم التعليمة الأولى ويحجز أربعة أماكن لتخزين العناصر الأربعة للمصفوفة A وهي : A[0], A[1], A[2], A[3]

تعليمة الحلقة for تؤدي إلى تكرار تنفيذ تعليمة الإدخال << cin خمس مرات من أجل قيم j تساوي 0 ,1 , 2 , 3 , 4 . وبالتالي ستقبل إدخال القيم الخمسة وتخزن القيم الأربعة الأولى من عناصر المصفوفة أما القيمة الخامسة فتخزن بعد العنصر الرابع مباشرة. أي أن المترجم لن يعطي أية رسالة خطأ رغم تجاوز حدود المصفوفة المصرح عنها في تعليمة التصريح float A[4].

مثال (3) : البرنامج التالي يصرح عن مصفوفة حقيقية ذات أربعة عناصر ثم يطلب حساب مجموع 5 عناصر من المصفوفة المكونة من 4 عناصر فقط.

```
# include < iostream.h >
main ()
{
      float A[4] = {2.5, 4.6, 6.3, 8.1};
      double sum = 0.0;
      for (int i = 0; i < 5; i++)
          sum = sum + A[i];      // or sum+= A[i];
      cout << " the sum of 5 elements is " << sum << endl;
      return 0;
}
```

المترجم سيقبل التنفيذ ويكون الخرج هو:

```
The sum of 5 elements is 2.50342 e + 11
```

المصفوفة المصرح عنها ذات أربعة عناصر فقط وعندما زاد دليل الحلقة i عن القيمة 3 وهو دليل العنصر الرابع وأصبح 4 فإن العنصر الخامس A[4] في هذه الحالة هو محتوى خلية في الذاكرة ليست ضمن عناصر المصفوفة ومحتوياتها غير معروفة أو متوقعة.

مجموع عناصر المصفوفة هو 21.5 ولكن المترجم تابع بالتنفيذ وجمع العنصر الخامس والذي يقع خارج حدود المصفوفة وأعطى الناتج 2.50342e+11.

إذا تم الإعلان عن المصفوفة من دون إعطاء قيمٍ ابتدائية لعناصرها وطلب تنفيذ تعليمة إظهار لتلك العناصر بعد تعليمة التصريح فإن جميع تلك العناصر تأخذ قيماً غير متوقعة أو عشوائية كما في المثال التالي :

مثال (4) : البرنامج التالي يصرح عن مصفوفة حقيقية ذات أربعة عناصر ثم يطلب طباعة قيم تلك العناصر.

```
# include < iostream.h >
main ()
{
       double A[4] ;
       for (int i = 0; i < 4; i++)
            cout << " A ["<< i << " ] = " << A [i]  << endl;
       return 0;
}
```

أدى البرنامج السابق إلى طباعة قيم مواقع الذاكرة التي عينت للمصفوفة A . وهي قيم عشوائية مخزنة سابقاً في تلك المواقع.

```
A[0] = 8,57724e-311
A[1] = 1.97412e-310
A[2] = 1.30807e-130
A[3] = 6.41597e-124
```

مثال (5): البرنامج التالي يعين قيماً لعناصر المصفوفة n الخمسة وفقاً للعلاقة :

قيمة العنصر = 2 + 2 × دليل العنصر

ويطبع بعد ذلك المصفوفة على شكل جدول.

```
// initialize array elements to even integers from 2 to 10
# include < iostream.h >
# include < iomanip.h >
main ()
{
        const int array_size = 5;
        int n[array_size ];
        for (int i = 0; i < array_size; i++)
            n[i] = 2 + 2 * i;
        cout << " element " << setw(7) << " value " <<endl;
        for ( i = 0; i < array_size; i++)
            cout << setw(7) << i << setw(7) << n[i] << endl;
        return 0;
}
```

والخرج هو :

```
Element   value
      0       2
      1       4
      2       6
      3       8
      4      10
```

يفيد استخدام اسم ثابت كحجم للمصفوفة في تغيير الحجم بسهولة. فبدلاً من 5 في المثال السابق يمكن أن نغير قيمته مثلاً إلى 1000 من أجل تنفيذه لمصفوفة ذات حجم مساوٍ لـ 1000 بكل سهولة وبساطة. ولولا ذلك لتطلب الأمر تنفيذ تعديلات في ثلاثة أماكن من البرنامج السابق.

مثال (6): البرنامج التالي يحسب مجموع قيم عناصر مصفوفة أحادية البعد S ويطبعه.

```
// compute the sum of elements of the array S.
# include < iostream.h >
main ()
{
     const int array_size = 6;
     int S[array_size] = { 1,3,5,4,7,2};
     int sum = 0;
     for (int j = 0; j < array_size; j++)
          sum = sum + S[j];
```

```
        cout << " total of array elements values is"
                << sum << endl;
        return 0;
}
```

والخرج هو :

```
total of array elements values is 22
```

يقوم البرنامج السابق بتعيين قيم ابتدائية لعناصر المصفوفة S في تعليمة التصريح ومن ثم يحسب مجموع تلك القيم بوساطة الحلقة for . يمكن إعطاء قيم أولية لعناصر المصفوفة S كقيم دخل معطاة من قبل المستخدم بوساطة لوحة المفاتيح كأن نكتب مثلاً :

```
for (int j = 0; j < array_size; j++)
        cin >> S[j];
```

مما يؤدي إلى قراءة قيمة مدخلة من لوحة المفاتيح في كل مرة تنفذ فيها الحلقة وتخزين تلك القيمة في العنصر S[j] . نستخدم هذه الطريقة فيما لو كانت قيم عناصر المصفوفة مختلفة في كل تنفيذ للبرنامج.

مثال (7): البرنامج التالي يقوم بقراءة 10 قيم لعناصر المصفوفة S ويظهرها في جدول أعمدته هي رقم العنصر ثم قيمته ثم عدد من النجوم يساوي قيمة ذلك العنصر كما في الشكل التالي:

element No.	element value	** S[i] times
0	2	**
1	3	***
2	5	*****

```
# include < iostream.h >
# include < iomanip.h >
main ()
```

```
{
        const int array_size = 10;
        int S [array_size ] = { 2,3,5,6,8,9,11,12,14,15};
        cout << " element No"<< "      "<< "element value"
              << "      "<<"** S[i] times " << endl;
        for (int i = 0; i < array_size; i++)
        {
              cout << setw(5) << i << setw(15) << S[i] << “    “;
              for (int j=1; j < = S[i]; j++)
                    cout << ' * ';
              cout << endl;
        }
        return 0;
}
```

والخرج هو :

```
element No.   element value         ** S[i] times
     0              2               **
     1              3               ***
     2              5               *****
     3              6               ******
     4              8               ********
     5              9               *********
     6             11               ***********
     7             12               ************
     8             14               **************
     9             15               ***************
```

7-3-3- فرز وترتيب المصفوفات (Array sort):

غالباً ما تكون قيم المتغيرات في الحياة العملية غير مرتبة بالصورة التي يمكن للحاسوب أن يتعامل معها. فمثلاً كثيراً مانحتاج إلى ترتيب الأسماء هجائياً كما في دليل

الهاتف أو تصنيف الناس حسب أعمارهم أو أطوالهم بغية أن يؤدي الحاسوب دوره في المعالجة بأكفأ أسلوب . هنالك عدة خوارزميات لفرز وترتيب العناصر.

المثال التالي يقوم بترتيب عناصر مصفوفة تصاعدياً حسب خوارزمية الترتيب الفقاعي.

مثال (8): تعتمد خوارزمية الترتيب الفقاعي Bubble sort على القيام بأكثر من مرور على عناصر المصفوفة، وفي كل مرور، تتم مقارنة عنصرين متتاليين، وإذا كانا مرتبين تصاعدياً أو لهما القيمة نفسها ، فإننا نتركهما على حالهما. أما إذا كانا غير مرتبين تصاعدياً، فإننا نقوم بالمبادلة بينهما ضمن المصفوفة، أي ننقل العنصر الأكبر خطوة إلى الأمام.

```
# include < iostream.h >
main ()
{
    const int array_size = 6,
    int A [array_size ] = { 10, 8, 4, 6, 1, 2 };
    int temp;
    cout << " Data items in original order : \ n ";
    for (int i = 0;  i < array_size, i++)
        cout << A[i] << "       ";
    for (int pass = 1; pass < array_size; pass++)
        for (int i = 0; i < array_size-1; i++)
            if (A[i] > A [i+1])
            {
                temp = A[i];
                A[i] = A[i+1];
                A[i+1] = temp;
            }
```

```
          cout << endl << " Data items in ascending order : \ n" ;
               for (i = 0;  i < array_size; i++)
                   cout << A[i] << "      ";
     return 0;
}
```

خرج البرنامج هو :

```
Data items in original order:
10   8   4   6   1   2
Data items in ascending order:
1   2   4   6   8   10
```

يقوم البرنامج بمقارنة A[0] مع A[1] ثم A[1] مع A[2] ثم A[2] مع A[3] حتى نصل إلى نهاية المصفوفة بمقارنة A[4] مع A[5] ، وبما أن المصفوفة تحوي ستة عناصر فالبرنامج يقوم بتنفيذ 5 مقارنات. وبما أن هذه المقارنات تتم بين العناصر المتتالية فإن القيمة الكبرى تشق طريقها إلى نهاية المصفوفة متجاوزة عدة أماكن بينما تقوم القيمة الصغرى بالتقدم باتجاه بداية المصفوفة مكاناً واحداً فقط.

من المضمون أن القيمة الكبرى تصل إلى نهاية المصفوفة بعد نهاية المرور الأول من أجل pass = 1 وفي نهاية المرور الثاني pass = 2 فإن القيمة الكبرى الثانية سوف تصل إلى الموضع 4 وهكذا حتى ينتهي تنفيذ الحلقتين المتداخلتين بإنهاء المرور الخامس.

تعُد هذه الخوارزمية بسيطة ولكنها ليست فعالة كغيرها من الخوارزميات المعروفة في ترتيب المصفوفات الجدول (7-1) يبين التغيرات من أجل كل مرور.

الجدول (7-1) التغييرات من أجل كل مرور في خوارزمية الترتيب الفقاعي

Pass	i	A[0]	A[1]	A[2]	A[3]	A[4]	A[5]
1	0	8	10	4	6	1	2
	1	8	4	10	6	1	2
	2	8	4	6	10	1	2
	3	8	4	6	1	10	2
	4	8	4	6	1	2	10
2	0	4	8	6	1	2	10
	1	4	6	8	1	2	10
	2	4	6	1	8	2	10
	3	4	6	1	2	8	10
	4	No change					
3	0	4	6	1	2	8	10
	1	4	1	6	2	8	10
	2	4	1	2	6	8	10
	3	No change					
	4	No change					
4	0	1	4	2	6	8	10
	1	1	2	4	6	8	10
	2	No change					
	3						
	4						
5	0	No change					
	1						
	2						
	3						
	4						

7-3-4- البحث في المصفوفة (Search in array):

تستخدم الحواسيب في العادة لغرض تخزين واستعادة البيانات بشكل واسع. وبما أن البيانات تخزن في مصفوفات. لذا عند البحث عن قيمة محددة (search – key) في المصفوفة. فإن أبسط طرق البحث المستخدمة تبدأ بمقارنة تلك القيمة مع كل عنصر من عناصر المصفوفة من بدايتها الواحد تلو الآخر، حتى يتم

العثور على الهدف المطلوب، في هـذه الطريقـة يكـون احتمـال وجـود القيمـة في بدايـة المصفوفة مساوياً لاحتمال وجودها في نهايتها.

تسمى هذه الطريقة بطريقة البحث الخطي (linear_search) وهي تعمل بشكل جيد مع المصفوفات صغيرة الحجم ولكنها غير فعالة مع المصفوفات كبيرة الحجم.

مثال (9): البرنامج التالي يستخدم طريقة البحث الخطي في البحث عن قيمة ضمن مصفوفة صحيحة مكونة من 8 عناصر. ويطبع عبارة مناسبة في كل حالة. ينتهي تنفيذ البرنامج لعدم تكرار البحث عن قيمة أخرى بإدخال قيمة مساوية 1-.

```
# include < iostream.h >
main ()
{
        const int size = 8;
        int B [ size ], search_key;
        int found;
        cout << " assign values for array elements:"<< endl;
        for (int j = 0;  j < size; j++)
              B[j] = 3 * j + 1;
        for ( j = 0; j < size; j++)
              cout << B[j] << "        ";
        cout << " \ n enter integer search_key, -1 to end:";
        while(1)
        {
              found = 0;
              cin >> search_key;
              if (search_key = = -1)
                   break;
              for (j = 0; j < size; j++)
                   if (B[j] = = search_key)
                        {
                             found = 1;
                             Cout << " search_key found at :    " << j << " \ n ";
                        }
        if  ( ! found )
            cout << " search_key not found \ n ";
        cout << " enter another value : ";
    }
    return 0;
}
```

خرج البرنامج هو :

```
assign values for array elements:
1   4   7   10   13   16   19   22
enter integer search_key, -1 to end:12↵
search_key not found
enter another value: 16↵
search_key found at: 5
enter another value: -1↵
```

4-7- المصفوفة ثنائية البعد (Two_dimensional array):

يمكن التعامل مع المصفوفة ثنائية البعد تماماً كالمصفوفة أحادية البعد.

نسمي المصفوفة ثنائية البعد إذا كان لكل عنصر من عناصرها دليلان بدل دليل واحد في حالة المصفوفة أحادية البعد. حيث يدل الدليل الأول على عدد الأسطر والدليل الثاني على عدد الأعمدة. وتتوضع قيم المصفوفة في نقاط تقاطع الأسطر مع الأعمدة. وتمثل جبرياً كالتالي:

$$a_{3,4} = \begin{bmatrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} \end{bmatrix}$$

لو أردنا تمثيل علامات مجموعة من الطلاب في عدد من المواد بشكل مصفوفة ثنائية. بحيث يمثل كل سطر العلامات التي حصل عليها الطالب الواحد في المواد المدروسة.

ويمثل كل عمود العلامات التي حصل عليها مجموعة الطلاب في مادة معينة. وبحيث يمثل تقاطع السطر مع العمود العلامة التي حصل عليها أحد الطلاب في إحدى المواد. لأمكننا كتابة تعليمة التصريح عن المصفوفة ثنائية البعد السابقة كالتالي:

```
const int no_of_subjects = 5;
const int no_of_students = 30;
float marks [ no_of_students ] [ no_of _subjects ];
```

التعليمة الأولى تصرح عن ثابت صحيح يمثل عدد الأعمدة no_of_subjects ومساوٍ لـ 5 وهو عدد المواد.

التعليمة الثانية تصرح عن ثابت صحيح يمثل عدد الأسطر no_of_students ومساوٍ لـ 30 وهو عدد الطلاب.

التعليمة الثالثة تصرح عن مصفوفة marks عناصرها قيم حقيقية وذات بعدين أسطرها تمثل عدد الطلاب وهو ثلاثون وأعمدتها تمثل عدد المواد وهي خمس.

7-4-1- إعطاء قيم ابتدائية لعناصر مصفوفة ثنائية البعد :

يتم تعيين قيم ابتدائية لعناصر المصفوفة ثنائية البعد تماماً كما هو الحال بالنسبة لمصفوفة أحادية البعد في تعليمة التصريح كالتالي:

```
int x[3] [4] = { 1,2,3,4,5,6,7,8,9,10,11,12};
```

حيث يتم الفصل بين القيم بالفاصلة العادية (,). ويتم حصر القائمة بالحاصرتين الكبيرتين { } بعد وضع إشارة التعيين (=) . تنتهي التعليمة طبعاً بالفاصلة المنقوطة (;) . يتم ذكر قيم العناصر ضمن اللائحة حسب ترتيب الأسطر، أي تذكر عناصر السطر الأول ثم الثاني وهكذا.

يمكن كتابة تعليمة التصريح السابقة بشكل أكثر وضوحاً كالتالي:

```
int x[3] [4] = {{1,2,3,4}, {5,6,7,8},{9,10,11,12}};
```

حيث تضمن عناصر كل سطر ضمن الحاصرتين الكبيرتين { }. وتفصل الفاصلة العادية (,) بين عناصر كل سطر.

لو كتبنا تعليمة التصريح التالية :

```
int x[3] [4] = {{1,2,3} , {5} , };
```

لكان ذلك مطابقاً للتعليمة التالية :

```
int x[3] [4] = {{ 1,2,3,0} , {5,0,0,0} , {0,0,0,0}};
```

حيث أن العناصر التي لاتعطى قيمٌ لها بشكل صريح تأخذ القيمة 0 ضمنياً.

7-4-2- إدخال وإخراج عناصر المصفوفة ثنائية البعد :

يمكن تطبيق الفقرة (7-3-2) على المصفوفات ثنائية البعد أيضاً ، أي يسمح لكل دليل أن يأخذ اسم متغير صحيح. كما يمكن استخدام تعليمة الإدخال cin << وتعليمة الإخراج cout << مع حلقتي تكرار متداخلتين وذلك لإدخال القيم من لوحة المفاتيح وإخراجها على الشاشة كما يلي:

```
int x[3][4];
for (int i = 0; i < 3; i++)
      {
            for (int j = 0;  j< 4; j++)
            {
                cin >> x[i][j];
                cout << x[i][j] << "      ";
            }
      cout << endl;
      }
```

التعليمات السابقة تقوم بالتصريح عن مصفوفة ثنائية صحيحة x[3][4] ذات ثلاثة أسطر وأربعة أعمدة، ثم تدخل قيماً وتطبعها على شكل مصفوفة بتنفيذ حلقتي التكرار المتداخلتين. حيث الحلقة الخارجية يتغير دليلها ليدل على أسطر المصفوفة الثلاثة بينما الحلقة الداخلية يتغير دليلها ليدل على أعمدتها الأربعة. عند الانتهاء من تنفيذ

الحلقة الداخلية لأول مرة نكون قد أدخلنا قيم عناصر السطر الأول وطباعته على الشاشة، ثم يأتي أمر الإخراج cout << endl; لينتقل إلى سطر جديد ويعود ويكرر طلب إدخال قيم عناصر السطر الثاني ومن ثم إظهاره على الشاشة وهكذا.

مثال (10): البرنامج التالي يدخل قيماً لعناصر مصفوفة ثنائية صحيحة x[3][4] ومن ثم يطبع تلك القيم على الشاشة.

```
# include < iostream.h >
# include < iomanip.h >
main ()
{
        int x [3] [4];
        for (int i =0; i < 3; i++)
        {
              for (int j =0; j < 4; j++)
                     {
                          cin >> x[i] [j];
                          cout << setw(4) << x[i] [j];
                     }
              cout << endl;
        }
        return 0;
}
```

خرج البرنامج هو :

```
1   2   3   4   5   6   7   8   9   10  11  12↵
    1   2   3   4
    5   6   7   8
    9   10  11  12
```

مثال (11) : اكتب برنامجاً يقوم بما يلي:

1) يدخل مصفوفة مربعة من 4 أسطر و 4 أعمدة.

2) يقوم بطباعتها سطراً سطراً.

3) طباعة عناصر القطر الرئيسي ومجموعها.

4) طباعة عناصر القطر الثانوي ومجموعها.

5) طباعة مجموع عناصر المصفوفة.

```
# include < iostream.h >
# include < iomanip.h >
main ()
{
      int B[4] [4];
      int summd = 0, sumsd = 0, sumB = 0;
      int i,j;
      cout << " enter array elements : " << endl;
      for (i = 0; i < 4; i++)
          for (j = 0; j < 4; j++)
              cin >> B[i] [j];
      cout << "print array B " << endl;
      for (i = 0; i < 4; i++)
      {
          for (j = 0; j < 4; j++)
              cout << setw(5) << B[i] [j];
              cout << endl;
      }
      cout < " the elements of the main diameter are : " << endl;
      for ( i = 0; i < 4; i++)
      {
          for ( j = 0; j < 4; j++)
              if ( i = = j)
              {
                  cout << B[i] [j] << "   ";
                  summd + = B[i] [j];
              }
          }
cout << " \ n " <<" and their sum is     "<< summd << endl;
cout << " the elements of the second diameter are:" << endl ;
for ( i = 0; i < 4; i++)
       {
            for (j = 0; j < 4, j++)
                 if ( i +j = = 3)
                 {
                      cout << B[i] [j] << "    ";
                      sumsd + = B[i] [j];
                 }
       }
cout << " \ n " << " and their sum is " << sumsd << endl;
for ( i = 0; i < 4; i++)
       for ( j = 0; j < 4; j++)
            sumB + = B[i] [j];
cout << " the sum of the array elements = " << sumB << endl;
return 0;
}
```

وخرج البرنامج :

```
enter array elements
1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16↵
print array B
1       2       3       4
5       6       7       8
9       10      11      12
13      14      15      16
the elements of the main diameter are:
1     6     11     16
and their sum is 34
the elements of the second diameter are:
4     7     10     13
and their sum is 34
the sum of the array elements = 136
```

7-5- السلاسل (Strings):

تُعد لغة ++C السلسلة الحرفية بمثابة مصفوفة من المحارف تخزن في مواقع متجاورة في الذاكرة تنتهي دوماً بالحرف الصفري ('0\')NULL . يتم التعامل مع السلاسل بوساطة متغيرات من النوع char . إن استعمال قيمة خاصة للإشارة إلى نهاية السلسلة يعني عدم الحاجة إلى تخزين طول السلسلة في متغير عددي صحيح منفصل. وعلى كل تابع يعالج السلاسل الحرفية أن يبحث عن حرف الإنهاء وهو ' 0\ ' من أجل معرفة مكان انتهاء السلسلة.

7-5-1- التصريح عن السلاسل الحرفية:

تعليمة التصريح التالية :

```
char  str [6];
```

تصرح عن سلسلة محرفية تستطيع تخزين سلسلة محارف تصل حتى 6 حروف بما فيها حرف النهاية '0\'. وليس من الضروري إعطاء قيمة ابتدائية للسلسلة في تعليمة التصريح.

7-5-2- إعطاء قيم ابتدائية للسلاسل المحرفية:

يمكن إعطاء قيم ابتدائية لسلاسل الحروف كما في التصريح التالي :

```
char str [   ] = "first";
```

حيث تعين قيمة ابتدائية لكل عنصر من عناصر السلسلة str تساوي أحد حروف السلسلة "first" . يتحدد عدد عناصر السلسلة str بوساطة المترجم وذلك حسب طول السلسلة المعطاة زائد واحد وهو للحرف الخاص '0\' الذي يحدد نهاية السلسلة، وبذلك تكون السلسلة str مكونة من ستة عناصر خمسة منها للقيمة "first" والسادس للحرف الصفري '0\'.

كما يمكن أيضاً إعطاء قيم ابتدائية لسلسلة الحروف باستخدام ثوابت الحروف المفردة ضمن قائمة للقيم الابتدائية. حيث يمكن كتابة التصريح السابق للسلسلة str كما يلي :

```
char str [   ] = { ' f ', ' i ', ' r ' , ' s ', ' t ', ' \ 0 ' };
```

لاحظ ضرورة وضع الحرف الصفري كآخر حرف في السلسلة المحرفية str في هذه الحالة. وعلى اعتبار أن سلاسل الحروف هي عبارة عن مصفوفات للحروف فيمكن الوصول إلى كل حرف من حروفها باستخدام دليل عناصر المصفوفة أي يمكن كتابة str[0]= ' f ' والعنصر ' r ' = [2] str ... والعنصر السادس والأخير هو '0\'=[5]str.

7-5-3- إدخال وإخراج قيم عناصر السلسلة المحرفية:

تعامل ++C السلاسل كنوع بيانات أساسي مثل int . والتعليمتان <<cout,>>cin تستخدمان لإدخال وإخراج السلسلة.

التعليمات التالية تنشئ متغيراً سلسلياً وتقرأ قيمته من لوحة المفاتيح ثم تعرض تلك القيمة على الشاشة.

```
char str [70];  //  creat a string variable str
cin >> str;      //   get text from user, store in str
cout << str;  //    display text enterd by user on screen
```

تعليمة الإدخال :

```
cin >> str;
```

تقبل من لوحة المفاتيح سلسلة من الحروف وتخزنها في str . لاحظ أنه تم كتابة اسم السلسلة المحرفية فقط في تعليمة الإدخال بوساطة << cin ولم تعط أية معلومة خاصة بعدد عناصرها. وبالتالي تقع على المبرمج مسؤولية التأكد من أن حجم السلسلة قادر على استيعاب سلسلة الحروف المدخلة من قبل المستخدم بوساطة لوحة المفاتيح.

يقوم << cin بقراءة الحروف المدخلة بوساطة لوحة المفاتيح حتى يصل إلى فراغ، أو حرف جدولة tab ، أو حرف سطر جديد " n \ " ، أو حرف نهاية ملف. أي أن

المترجم لايهتم بحجم السلسلة وبالتالي يمكن له أن يقبل إدخال معطيات تتجاوز حجم السلسلة المصرح عنه في تعليمة التصريح.

يمكن استخدام التابع setw(N) للتأكد من أن طول السلسلة المدخلة إلى str لن يتجاوز حجمها. فالتعليمة التالية مثلاً :

```
cin >> setw (20) >> str;
```

تحدد لـ cin قراءة 19 حرفاً كحد أعلى وتخزنها في str وتخصص الحرف رقم 20 لتخزين الحرف الصفري ' 0 \'.

تؤمن لغة ++C التابع cin.getline الذي يأخذ ثلاثة باراميترات هي : اسم السلسلة التي نرغب بتخزين السطر فيها كبارامير أول. وطول هذه السلسلة كبارامير ثاني. أما البارامير الثالث فهو الحرف الذي يحدد نهاية السطر. مثلاً التعليمات التالية:

```
char str[70];
cin.getline(str, 70, ' \ n ' );
```

تصرح عن سلسلة محرفية مؤلفة من 70 حرفاً واسمها str . ثم يقرأ سطراً مدخلاً بوساطة لوحة المفاتيح إلى السلسلة السابقة. ينهي التابع عمله عند قراءة الحرف 'n\'، أو عندما يصادف حرف نهاية ملف، أو عندما يتجاوز عدد الحروف المدخلة قيمة البارامير الثاني له منقوصاً منه العدد واحد (حيث يتم تخصيص الحرف الأخير لتخزين الحرف الصفري '0\' كما ذكرنا سابقاً).

إذا تم إدخال الحرف الذي يحدد نهاية السطر، فتتم قراءته ولايخزن. تُعد قيمة البارامير الثالث الافتراضية '0\' . أي يمكن كتابة التعليمة من دونه على الشكل التالي:

```
cin.getline(str,70);
```

يمكن أيضاً طبع السلسلة الممثلة كمصفوفة محرفية بوساطة التابع >> cout . أي يمكن طباعة السلسلة str بوساطة التعليمة التالية:

```
cout  << str ;
```

لاحظ أن << cout مثل >> cin لاتهتم لحجم المصفوفة المحرفية حيث تتم طباعة حروف السلسلة حتى الوصول إلى الحرف الصفري (أي نهايتها).

يمكن إدخال وإخراج السلسلة المحرفية باستخدام التابعين cin.get() و cout.put() . والمعرفين في الملف الرأسي < iostream.h > حيث يستخدم التابع cin.get() بالشكل:

```
char st;
st = cin.get();
```

يقوم هذا التابع بقراءة الحرف المدخل من لوحة المفاتيح وتخزينه في المتغير المحرفي st . وبالتالي يمكن استغلاله لتخزين السلسلة التي نرغب في مصفوفة محرفية كما يلي:

```
do
{
        str = cin.get ( );
        strarray [n] = str;
        n++;
} while (str ! = ' \ n ' );
```

كذلك يستخدم التابع cout.put() بطباعة الحرف المخزن في المتغير المحرفي str على الشاشة . وبالتالي يمكن أيضاً إخراج السلسلة المحرفية كما يلي:

```
for (int i = 0; i < n; i++)
    cout. put (strarray[i]);
```

مثال (12) : البرنامج التالي يدخل نصاً ثم يحسب عدد مرات ورود الحرف ' e ' فيه :

```
# include < iostream.h >
main ()
{
        char ch;
        int count = 0;
        do
        {
            ch = cin.get ();
            if (ch = = ' e ')
                 count++,
        } while (ch ! = ' \ n ');
            cout << count << " e ' s were counted. \ n ";
        return 0;
}
```

الخرج هو كما يلي وحسب النص الوارد :

```
Treat people nice and gentel↵
6e's were counted
```

مثال (13) : البرنامج التالي يتعامل مع سلاسل محرفية ويبين كيفية القراءة من سلسلة حروف وطباعة السلسلة الناتجة وطريقة الوصول إلى كل حرف من حروف السلسلة المستخدمة.

```
// treating character arrays as strings
# include < iostream.h >
main (  )
{
       char  s1[25], s2[   ] = " C++ program ";
       cout << " enter a value for s1  : ";
       cin >> s1;
       cout << " s1 is : " << s1 << endl << " s2 is : " << s2 << endl
            << " s1 printed with spaces between its characters : " << endl;
       for (int i = 0; s1[i] ! = ' \ 0 ' ; i++)
            cout << s1[i] << "       ";
       cout << endl;
       return 0;
}
```

لو أدخلنا السلسلة المحرفية C++ programming language عند الاستجابة لتعليمة الإدخال cin >> s1; لكان الخرج كالتالي:

```
enter a value for s1 : C++ programming language↵
s1 is : C++
s2 is : C++ program
s1 printed with spaces between its characters:
C   +   +
```

لو أعدنا كتابة البرنامج السابق واستخدمنا التعليمة التالية :

```
cin.getline (s1 , 25 , ' \ n ' )
```

بدلاً من التعليمة ;cin >> s1 . وباستخدام نفس القيمة لـ s1 وهي C++ programming language لكان البرنامج كالتالي:

```
// treating character arrays as strings
# include < iostream.h >
main ()
{
      char  s1[25], s2[   ] = " C++ program ";
      cout << " enter a value for s1 : ";
      cin.getline (s1,25,'\n');
      cout << " s1 is : " << s1 << endl << " s2 is : " << s2 << endl
          << " s1 printed with spaces between its characters : " << endl;
      for (int i = 0; s1[i] ! = ' \ 0 ' ; i++)
          cout << s1[i] << "        ";
      cout << endl;
      return 0;
}
```

والخرج هو :

```
enter a value for s1 : C++ programming language↵
s1 is : C++ programming language
s2 is : C++ program
s1 printed with spaces between its characters:
c + +   p r o g r a m m i n g   l a n g u a g e
```

مثال (14) : البرنامج التالي يقبل إدخال محرفاً من لوحة المفاتيح ويعينه لسلسلة محرفية word. ينهي عملية الإدخال بإدخال المحرف ' \ n ' .

```
# include < iostream.h >
```

```
main ()
{
     char str [50];
     int ch, n = 0;
     cout << " enter a text, press new line key to end: " << endl;
     do
     {
          ch = cin.get(  );
          str [n] = ch;
          n++;
     } while ( ch! = ' \ n ');
          cout << " the text entered is : " << endl;
          for ( int i = 0; i < n; i++)
               cout << str [i]; //or cout.put (str [i]);
     return 0;
}
```

والخرج هو :

```
enter a text, press new line key to end:
C++ programming language↵
the text entered is:
C++ programming language
```

6-7- مصفوفات السلاسل (String arrays):

السلاسل التي تعرضنا لها حتى الآن هي مصفوفات محرفية أحادية البعد. أما المصفوفة ثنائية البعد فهي في الحقيقة مصفوفة أحادية البعد ولكن كل عنصر من عناصرها عبارة عن مصفوفة محرفية أخرى أحادية البعد. ويتم الإعلان عن مصفوفة السلاسل كالتالي:

```
char   string_array_name [size1] [size2];
```

مثال :

```
char   color [5] [8];
```

يعلن عن مصفوفة سلاسل color مؤلفة من خمسة أسطر وكل سطر يتضمن سلسلة لايزيد طول محارفها عن 8 محارف. نصل إلى هذه المحارف من خلال color [0] و color[1] ... color[4] ويمكن إعطاء قيمة ابتدائية لمصفوفة السلاسل باستخدام قائمة التخصيص للقيم الابتدائية.

7-7- مكتبة توابع التعامل مع سلاسل المحارف

(string manipulation function of the string handling library):

توفر لغة c++ مجموعة من التوابع التي تسهل عملية معالجة السلاسل ، ولابد من أن يحتوي البرنامج على التوجيه التالي عند استخدام أي من هذه التوابع.

```
# include < string.h >
```

وذلك لتضمين الملف الرأسي < string.h > المعرفة فيه هذه التوابع.

سنبين فيما يلي بعض هذه التوابع:

7-7-1- التابع strlen (S)

يعيد التابع strlen طول السلسلة المحرفية S أي عدد المحارف المشكلة للسلسلة المحرفية وبدون رمز النهاية ' 0 \'.

مثال (15) : البرنامج التالي يدخل سلسلة محرفية ثم يقوم بحساب عدد محارفها.

```
# include < iostream.h >
# include < string.h >
main ()
{
        char s1[30], s2[30];
        cout << " enter the first string s1 : " << endl;
        cin.getline (s1, 30, ' \ n ');
        cout << " enter the second string s2 : " << endl;
        cin.getline (s2, 30, ' \ n ');
        cout << "s1 = " << s1 <<  endl;
        cout << " s2 = " << s2 << endl;
        cout << " the length of s1 is : " << strlen (s1) << endl;
        cout << " the length of s2 is : " << strlen (s2) << endl;
        return 0;
}
```

والخرج هو :

```
enter the first string s1:
Aleppo university⏎
enter the second string s2:
Communication Department⏎
S1 = Aleppo university
S2 = Communication Department
the length of s1 is : 17
the length of s2 is : 24
```

7-7-2- التابع strcmp (s1,s2) و strncmp(s1,s2,n)

يقوم التابع strcmp بمقارنة سلسلتين محرفيتين محرفاً محرفاً على التوالي ويعيد القيمة 0 إذا تساوت السلسلتان، بينما يعيد قيمة سالبة إذا كانت السلسلة الأولى (البارامیتر الأول) أصغر من السلسلة الثانية (البارامیتر الثاني). أو قيمة موجبة إذا كانت السلسلة الأولى أكبر من السلسلة الثانية، والقيمة الموجبة أو السالبة هي الفرق بين قيمتي المحرفين الصحيحة بشيفرة ASCII.

أما التابع strncmp فيقوم بمقارنة فقط n رمز أو محرف من السلسلة الثانية مع n محرف من الأولى محرفاً محرفاً على التوالي وتكون النتيجة هي أيضاً واحدة من الحالات الثلاث التالية:

1- قيمة مساوية للصفر إذا تساوى n محرف من السلسلة الأولى s1 مع n محرف من السلسلة s2 .

2- قيمة أكبر من الصفر إذا كانت نتيجة مقارنة n محرف من s1 أكبر من n1 محرف من s2.

3- قيمة أصغر من الصفر إذا كانت نتيجة مقارنة n محرف من s1 أصغر من n محرف من s2.

مثال (16): البرنامج التالي يوضح استخدام التابعين ( ) strcmp و strncmp( )

```
# include < iostream.h >
# include < string.h >
main ()
{
    char s1[   ] = " My computer";
    char s2[   ] = " My comfort";
    char s3[20], s4[20];
    cout << " strcmp(s1,s2) = " << strcmp (s1,s2) << endl;
    cout << " enter string s3 : " << endl;
    cin.getline(s3,20, '\n');
    cout << " enter string s4:"<< endl;
    cin.getline(s4,20, '\n');
    cout <<< " s3="<<s3<<endl;
    cout<<" s4 = " << s4 << endl;
    cout << " strcmp (s3,s4)=" << strcmp (s3,s4) << endl;
    if (strcmp (s3,s4)= = 0)
     {
        cout << " strcmp (s3,s4) = = 0 ) so";
       cout << " s3 is the same as s4 " << endl;
    }
    if (strcmp(s3,s4) < 0 )
        cout << " strcmp (s3,s4) < 0 so ";
    cout << " s3 is less than s4 " << endl;
    }
    If (strcmp (s3,s4) > 0)
    {
```

```
    cout << " strcmp (s3,s4)> 0 so";
    cout << " s3 is greater than s4 " << endl;
   }
   cout << "strncmp(s3,s4,6)="<<strncmp(s3,s4,6) << endl;
   return 0;
}
```

والخرج هو :

```
strcmp (s1,s2) = 10
enter string s3:
Aleppo University ↵
enter string s4:
Aleppo citadel ↵
S3 = Aleppo University
S4 = Aleppo Citadel
strcmp (s3,s4) = -14
strcmp (s3,s4) <0 so s3 is less than s4
strncmp (s3,s4,6) = 0
```

7-3-7- التابع strcat (s1,s2) والتابع strncat(s1,s2,n)

يقوم التابع strcat بإلحاق السلسلة المحرفية s2 (البارامتر الثاني) في نهاية السلسلة المحرفية s1 (البارامتر الأول) والذي أعلن عنه كمصفوفة محرفية وبطول كاف لدمج السلسلتين.

أما التابع strncat فيقوم بإلحاق n محرف الأولى من السلسلة المحرفية s2 في نهاية السلسلة المحرفية s1 والتي أعلن عنها أيضاً بطول كاف لاستيعاب التغيير الناتج عن تنفيذ التابع. كما يتم إضافة المحرف الصفري للسلسلة الناتجة.

مثال (17) : البرنامج التالي يوضح استخدام التابعين strcat و strncat.

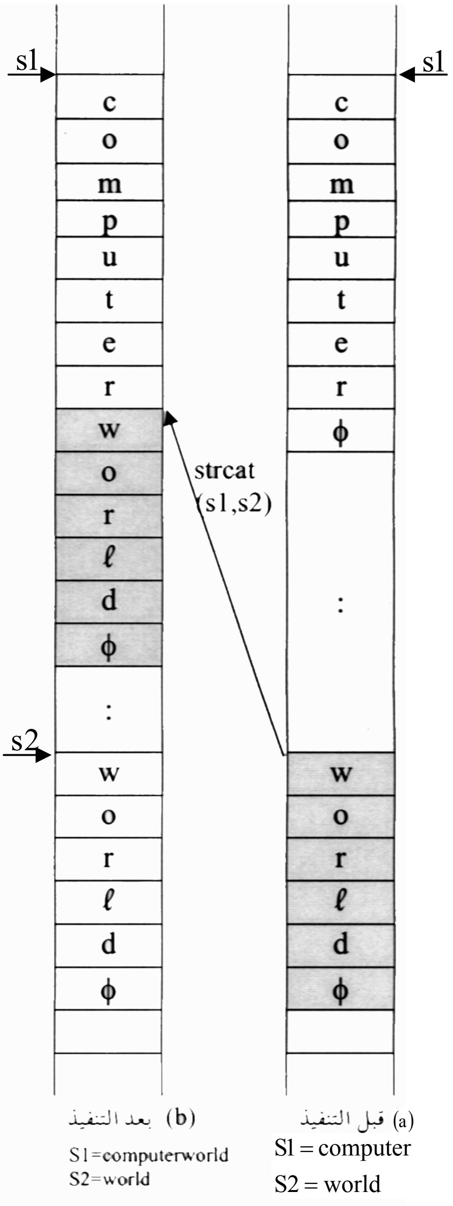
```
# include < iostream.h >
# include < string.h >
```

```
main ()
{
    char s1[30], s2[30];
    cout << " enter the first string s1 : " << endl;
    cin.getline(s1,30, '\n');
    cout << " enter the second string s2:"<< endl;
    cin.getline(s2,30, '\n');
    cout << " s1="<<s1<<endl<<"s2="<< s2<<endl;
    cout<<"strcat(s1,s2)="<<strcat(s1,s2)<<endl;
    cout<<"s1="<<s1<<endl;
    cout << " strncat (s1,s2,2) = "
          << strncat (s1,s2,2) << endl;
    cout << " s1 = " << s1 << endl;
    return 0;
}
```

والخرج هو :

```
enter the first string s1:
computer↵
enter the second string s2:
world↵
s1 = computer
s2 = world
strcat (s1,s2) = computerworld
s1 = computerworld
strncat (s1,s2,2) = computerworldwo
s1 = computerworldwo
```

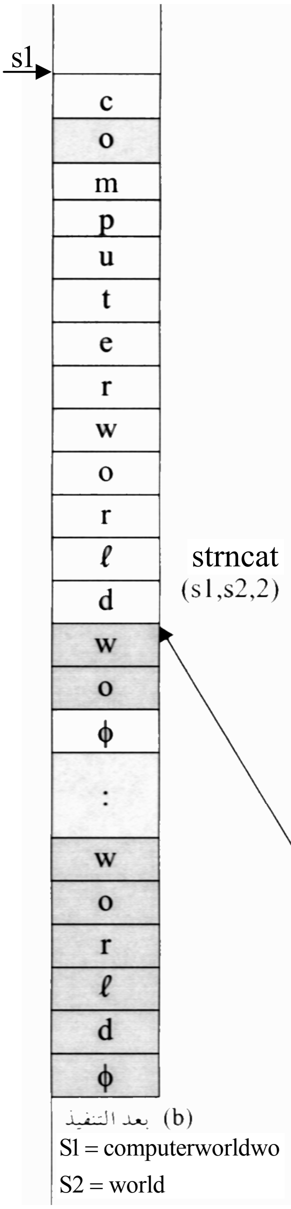
ويمكن تصور مواقع الذاكرة المخزنة لهاتين السلسلتين قبل تنفيذ التابع strcat وبعده كما في الشكل (7-1) وكذلك قبل تنفيذ التابع strncat وبعده كما في الشكل (2-7).



(a) قبل التنفيذ
S1 = computer
S2 = world

(b) بعد التنفيذ
S1=computerworld
S2=world

الشكل (7-1) **عمل التابع** strcat



(a) قبل التنفيذ
S1=computerworld
S2=world

(b) بعد التنفيذ
S1 = computerworldwo
S2 = world

الشكل (7-2) **عمل التابع** strncat

7-7-4- التابع Strcpy (s1,s2) والتابع Strncpy(s1,s2,n)

يقوم التابع بنسخ السلسلة المحرفية s2 إلى السلسلة المحرفية s1 التي أعلن عنها كسلسلة ذات طول مناسب لتخزين محارف السلسلة s2 إضافة للمحرف الصفري. أي

أن التابع strcpy(s1,s2) يكون نسخة مرادفة للسلسلة s2 والسلسلتان منفصلتان بحيث إن تغيير إحداهما ليس له تأثير على الأخرى.

التابع strncpy(s1,s2,n) يقوم بنسخ n محرف والأولى من s2 في s1 والتي أعلن عنها كسلسلة محرفية ذات طول كافٍ لتخزين n+1 محرف على الأقل مع الأخذ بالحسبان رمز النهاية.

مثال (18) : البرنامج التالي يوضح عمل التابع strcpy(s1,s2) و strncpy(s1,s2,n)

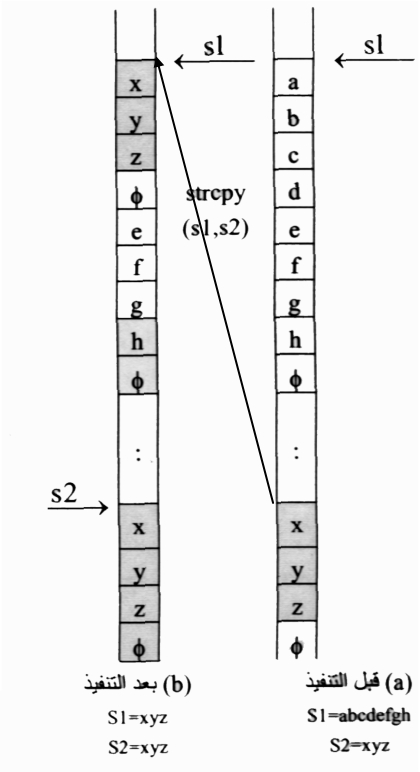
حيث s1 = abcdefg و s2 = xyz

```
# include < iostream.h >
# include < string.h >
main ()
{
    char s1[50], s2[50];
    int n;
    cout << " enter the first string s1:" << endl;
    cin.getline (s1,50, '\n');
    cout << " enter the second string s2: "<<endl;
    cin.getline (s2,50,'\n');
    cout << " s1 = " << s1 << endl;
    cout << " s2 = " << s2 << endl;
    strcpy (s1,s2);
    cout << " s1 after strcpy (s1,s2) is: "
        << endl << " s1 = " << s1 << endl;
    while(1)
    {
        cout << " enter n > 0 : and 0 to end : ";
        cin >> n;
        if (n = = 0) break;
        strncpy (s1,s2,n);
        cout << " s1 after strncpy (s1,s2,n) is : "
             << endl << " s1 = " << s1 << endl;
    }
    return 0;
}
```

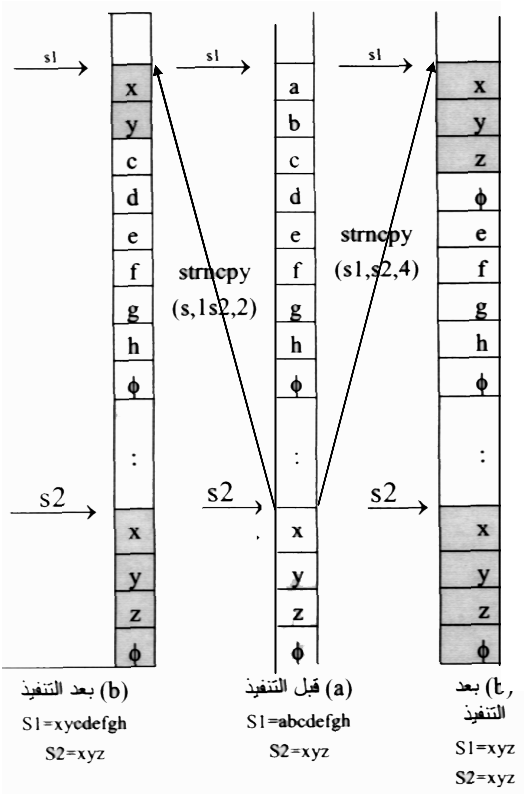
والخرج هو :

```
enter the first string s1:
abcdefgh↵
enter the second string s2:
xyz↵
s1 = abcdefgh
s2 = xyz
s1 after strcpy (s1,s2) is :
s1 = xyz
enter n > 0 : and 0 to end:2
s1 after strncpy (s1, s2,n) is:
s1 = xyz
enter n > 0 : and 0 to end : 4
s1 after strncpy (s1,s2,n) is :
s1 = xyz
enter n > 0 : and 0 to end : 0
```

ويمكن تصور مواقع التخزين في الذاكرة لكل من السلسلتين s1 و s2 قبل تنفيذ التابع strcpy وبعده كما في الشكل (7-3) . وقبل تنفيذ التابع strncpy وبعده كما في الشكل (7-4).



الشكل (7-3) عمل التابع strcpy



الشكل (7-4) عمل التابع strncpy

7-7-5- التابع strtok (s1,s2)

يستخدم التابع strtok لتقسيم سلسلة حروف إلى عدة واحدات (مقاطع tokens) يتألف كل منها من عدة حروف ويفصل بين كل مقطع وآخر حروف السلسلة s2 . نكرر استدعاء التابع strtok للحصول على كل كلمة من كلمات السطر في نص يفصل بين كلماته إما الفراغ أو حروف علامات الترقيم (delimiting characters).

يأخذ الاستدعاء الأول للتابع بارامترين ، الأول هو s1 السلسلة المطلوب تقطيعها أو تقسيمها إلى مقاطع ، أما البارامتر الثاني s2 فهو السلسلة التي تتضمن الحروف التي تفصل ما بين المقاطع (أي الحروف الفواصل).

ويحل البارامیتر NULL محل البارامیتر الأول s1 في الاستدعاءات التالية للاستدعاء الأول .

يعيد هذا التابع مؤشراً على بداية الوحدة التي يوجدها بعد كل استدعاء. ويعيد القيمة NULL في حال عدم وجود أية وحدة جديدة أو مقطع جديد.

مثال (19) : البرنامج التالي يوضح استخدام التابع strtok لاستخلاص الكلمات من جملة مدخلة.

```
# include < iostream.h >
# include < string.h >
main()
{
    char s[ ] = "today is the first day at school.";
    char * p;
    cout << " The string s is : [ " << s << " ] " << endl
         << " It’s tokens are : " << endl;
    p = strtok (s, " ");
    while (p)
    {
       cout << "\t" << p << endl;
       p = strtok (NULL, " ");
    }
    cout << " Now the string s is : " << s << endl;
  return 0;
}
```

خرج البرنامج هو :

```
The string s is :[ today is the first day at school.]
It‘s token are:
            today
```

```
        is
        the
        first
        day
        at
        school
Now the string s is : today
```

الاستدعاء الأول للتابع في التعليمة :

```
p = strtok (s, " ");
```

يحدد المؤشر ليشير إلى الجزء الأول من السلسلة s ، ويغير المسافة الخالية التي تلي today إلى الحرف الصفري NULL ('\0') (المشار له بالرمز ϕ في المخطط التالي). يؤدي ذلك إلى أن كلاً من s و p أصبح يساوي السلسلة today . بعد ذلك كل استدعاء تالٍ يأخذ الشكل :

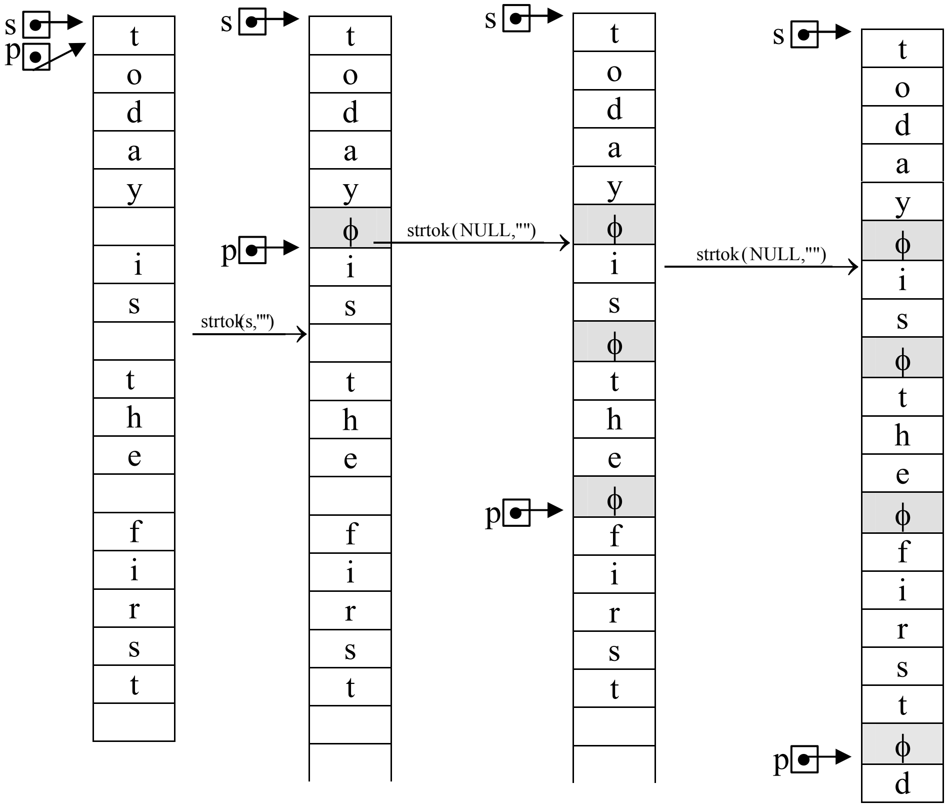
```
p = strtok (NULL, " ");
```

يقدم المؤشر p إلى حرف غير خالٍ تالٍ للحرف الصفري الجديد NULL.

إن تغيير كل مسافة خالية يمر عليها المؤشر إلى حرف صفري NULL . وتغيير أول حرف تالي p* إلى الحرف الصفري. ذلك له تأثير جعل p كسلسلة جزئية تالية للتي كانت تنتهي بمسافات والآن تنتهي بالحرف الصفري. يستمر ذلك حتى تصل p إلى الحرف NULL الذي ينهي السلسلة الأصلية s . هذا يجعل p = NULL أي الصفر ويوقف الحلقة while .

التأثير المجمل على السلسلة الأصلية s نتيجة كل النداءات لـ strtok ( ) هو تغيير كل مسافة خالية إلى NULL . هذا يجزئ السلسلة s. ويغيرها إلى تتابع من سلاسل جزئية محددة يكون الأول منها فقط معرفة بـ s. إن التابع strtok( ) يغير السلسلة لأنه يجزئها لهذا إذا أردنا استخدام السلسلة الأصلية بعد تجزئتها فلابدّ من

الاحتفاظ بنسخة منها باستخدام التابع strcpy( ). الشكل (7-5) التالي يوضح عمل التابع strtok().



الشكل (7-5) يوضح عمل التابع strtok( )

7-8- تمارين

1- اكتب برنامجاً يحسب الجداء التالي :

$$P = (y - x_1)(y - x_3)(y - x_5):...(y - x_n)$$

حيث n عدد فردي يدخل إلى البرنامج مع y وعناصر المصفوفة الأحادية x[i] ، (i=1,2,...,n)

خرج البرنامج قيمه p.

```
# include < iostream.h >
```

```
main( )
{
     int n,y,x[100], i,p=1;
     cout << " enter an odd integer number n:";
start: cin >> n;
     if (n%2 = = 0)
     goto start;
     cout << " enter an integer number y:";
     cin >> y;
     cout << " enter an integer elements for array x:";
     for (i=1; i<=n; i++)
        cin >> x[i];
     for (i=1; i<=n; i+=2)
        p=p*(y-x[i]);
     cout << "p=" << p << endl;
     return 0;
}
```

والخرج هو :

```
enter an odd integer number n: 9↵
enter an integer number y: 10↵
enter an integer elements for array x: 1 2  3 4 5 6 1 2 3 4↵
p = 19845
```

2- اكتب برنامجاً يسمح بإدخال عناصر مصفوفة أحادية S ذات N عنصراً صحيحاً. ومن ثم يوجد ويطبع العنصر الأصغر min والعنصر الأكبر max من عناصر المصفوفة والفرق بينهما diff.

```
# include <iostrean.h>
main( )
{
     int S[100];
     int N,min,max,diff;
     cout << "enter array's elements number:";
```

```
    cin >> N;
    cout << "enter array’s elements values: " << endl;
    for (int i=0; i<N; i++)
      cin >> S[i];
    max = min = s[0];
    for (i=1; i<N, i++)
      {
        if (S[i] > max)
            max = S[i];
        else if (S[i]<min)
            min = S[i];
      }
      diff = max - min;
      cout << "max = " << max << "\n"
          << "min = " << min << "\n"
          << "diff = " << diff << endl;
    return 0;
}
```

والخرج هو :

```
enter array’s elements number: 8↵
enter array’s elements values:
3   -4    7   12    -5    33    23    -21    23↵
Max = 33      min = -21       diff=54
```

3- اكتب برنامجاً يسمح بإدخال علامات N طالب في مادة البرمجة، ويخزنها في مصفوفة marks[N] ويختبر قيم هذه العلامات (يجب أن تكون العلامة بين الصفر والمئة) ويخرجها، ثم يحسب ويطبع عدد الطلاب الراسبين nfail ومتوسط علاماتهم avefail (علامة النجاح هي 48) وكذلك عدد الطلاب الناجحين npass ومتوسط علاماتهم avepass.

```
# include < iostream.h >
```

```
main ( )
{
    int marks[100];
    int n, nfail, sumf, npass, sump;
    int avefail, avepass;
    cout << "enter number of students:";
    cin >> n
    sumf = sump = 0;
    nfail = npass = 0;
    cout << "enter students marks:";
    for (int i = 0 ; i < n; i++)
       {
          cin >> marks [i];
              if (marks[i] > 0 && marks[i] < 100)
                   {
                      cout << marks[i]<<",";
                                if (marks[i] < 48)
                                 {nfail + = 1;
                                     sumf + = marks[i];
                                  }
                                else
                                     {npass + = 1;
                                     sump + = marks [i]
                                     }
                   }
             else
                continue;
          }
       cout << "sumf = " << sumf << " \t" <<"sump="<<sump<<endl;
       avefail = sumf/nfail;
       avepass=sump/npass;
       cout << "nfail="<<nfail<<"\t"<<"avefail="<<avefail<<endl;
       cout<<"npass="<<npass<<"\t"<<"avepass="<<avepass << endl;
    return 0;
}
```

والخرج هو :

```
enter number of students : 23↵
enter student marks : 5762 54 64 82 61 60 33 83 64 44 21 10 23 78 0↵
54 , 64 , 82 , 61, 33 , 83 , 64 , 44↵
sumf = 208        sump = 954
nfail = 7         avefail = 29
npass = 14        avepass = 68
```

4- اكتب برنامجاً لحساب مجموع كل من الأعداد الزوجية الموجبة sp وعددها Np وكذلك مجموع الأعداد الزوجية السالبة SN وعددها NN ، الواردة ضمن السلسلة A ذات N عنصر صحيح. دخل البرنامج عدد الأعداد وقيمها وخرج البرنامج هو قيم Sp ، Np ، SN ، NN .

```
# include < iostream.h >
main ()
{
    int sp, np, sn, nn;
    int A [100], n;
    cout << " enter number of elements in array A:";
    cint >> n;
    cout << " enter values of array elements:"<<endl;
    for (int i = 0; i < n; i++)
       cin >> A[i];
    sp = np = sn = nn = 0;
    int  j = 0;
    while ( j < n )
       {
          if (A[j] = = 0)
          goto loop;
          if ( A[j] %2 = = 0 && A[j] > 0 )
               {
                sp + = A[j];
                np + = 1;
            }
        if ( A[j] %2 = = 0 && A[j] < 0 )
            {
                sn + = A[j];
                nn + = 1;
            }
        loop : j++;
        }
            cout << "sp="<<sp<<"\t"<<"np="<<np<<"\n";
            cout <<"sn="<<sn<<"\t"<<"nn="<<nn<<endl;
    return 0;
}
```

والخرج هو :

```
enter number of elements in array A:8↵
enter values of array elements:
-1   2   7   -8   14   0   18   -4↵
sp = 34        np = 3
sn = -12       nn = 2
```

5- اكتب برنامجاً يسمح بإدخال N عدداً صحيحاً ويخزنها في المصفوفة A ثم يطبعها في عمود أول وفي العمود الثاني يطبع القيم وفقاً لما يلي:

مكعب الأعداد التي تحقق الشرط : $1 <= A[i] <= 10$

مربع الأعداد التي تحقق الشرط : $10 < A[i] <= 15$

وما عدا ذلك يخرج العبارة " Bad value "

```
# include < iostream.h >
# include < math.h >
main ( )
{
    int a[100], n;
```

```
      cout << " enter number of elements : "
      cin >> n;
      cout << " enter array elements values : " << endl;
      for ( int i = 0; i < n; i++ )
         {
            cin >> a[i];
            if (a[i] > = 1 && a[i] < = 10)
              cout << a[i]<< "\t" << pow (a[i] , 3) << endl;
            else  if (a[i] > 10 && a[i] < = 15)
                    cout << a[i] << "\t" << a[i] * a[i] << endl;
                   else
                    cout << a[i] << "\t" << " bad value " << endl;
         }
    return 0;
}
```

وخرج البرنامج هو :

```
enter number of elements : 8⏎
enter array elements values :
2   4   8   4   11   13   6   23   44   13⏎
2      8
4      64
8      512
4      64
11     121
13     169
6      216
23     bad value
```

6- اكتب برنامجاً يقرأ قيماً لعناصر المصفوفة A ذات عشر قيم حقيقية ثم يقرأ رقمين صحيحين I و J ويبدل قيم A[J] و A[I] ، إذا كانت A[J] أصغر من A[I] . ثم يطبع عناصر المصفوفة ثانية.

```
# include < iostream.h >
```

```
main ( )
{
     float a[10], temp;
     int i , j;
     cout << " enter array elements : " << endl;
     for (int k = 0; k < 10; k++)
        cin >> a[k];
     cout << " enter two integer values:" << endl;
     cin >> i >> j;
     if (a[j] < a[i])
        {
            temp = a[j];
            a[j] = a[i];
            a[i] = temp;
        }
     cout << " array elements after comparing a[i] and a[j]:" << endl;
        for (k=0; k<10; k++)
            cout << a[k] << ",";
        cout << endl;
     return 0;
}
```

والخرج هو :

```
enter array elements :
11   21   32   43   34   12   89   67   45   65↵
enter two integer values:
2   5↵
array elements after comparing a[i] and a[j] :
11 , 21 , 12 , 43 , 34 , 32 , 89 , 67 , 45 , 65
```

7- إذا أعطي عدد صحيح n1 فإنه يمكن تشكيل مصفوفة من الأعداد n1 ، n2 ، n3 ، ... ، nk ، وذلك وفقاً لما يلي :

1- إذا كان n_i عدداً فردياً يحسب n_{i+1} وفق العلاقة $n_{i+1} = n_i * 5 + 1$

2- إذا كان n_i عدداً زوجياً يحسب n_{i+1} وفق العلاقة : $n_{i+1} = n_i / 2$

3- تعين العناصر التالية بالطريقة نفسها . أي بالاعتماد على العنصر السابق.

اكتب برنامجاً يعين قيماً للعناصر العشرة الأولى للمصفوفة n المشكلة وفق الخطوات السابقة.

مثال: إذا كان العدد المدخل هو 1 ، تكون قيم عناصر المصفوفة هي:

1, 6, 3, 16, 8, 4, 2, 1, 6, 3

```
# include < iostream.h >
main ( )
{
    int num, n[10], i;
    cout << " enter an integer positive number:";
start: cin >> num;
      if (num < = 0)
          goto start;
      n[0] = num;
      for (i = 0; i < 10; i++)
       {
          if (n[i]%2 = = 0)
              n[i+1] = n[i]/2;
          else
              n[i+1] = n[i] * 5+1;
       }
    cout << "the array elements are:"<<endl;
    for (i = 0; i<10; i++)
       cout << n[i] << ",";
    cout << endl;
    return 0;
}
```

والخرج هو :

```
enter an integer positive number:5↵
```

```
the array elements are:
5 , 26 , 13 , 66 , 33 , 166 , 83 , 416 , 208 , 104
```

8- اكتب برنامجاً دخله عدد صحيح موجب n1 وخرجه عناصر المصفوفة المحسوبة وفق خطوات المثال (7) السابق بالنسبة للعدد الزوجي أما إذا كان العدد ni فردياً فإن ni+1 يحسب وفق العلاقة : n+1 = ni*3+1، وكذالك عدد العناصر k . تنتهي السلسلة عند مصادفة العدد واحد لأول مرة أي $(n_k = 1)$.

```
# include < iostream.h >
main ( )
{
    int num, n[100], i,k=0;
    cout << " enter an integer positive number:";
start: cin >> num;
    if (num < = 0)
    goto start;
    n[k] = num;
loop: if (n[k]%2 = = 0)
    n[k+1] = n[k]/2;
    else
    n[k+1] = n[k] * 3+1;
    k+=1;
    if (n[k] = = 1)
       {
         k+ = 1;
         cout << "number of array elements k = " << k << endl;
         goto end;
       }
    else
       go to loop;
end:  for (i = 0; i < k; i++)
         cout << n[i] << ",";
       cout << endl;
    return 0;
}
```

والخرج هو :

```
enter an integer positive number:3 ⏎
number of array elements k = 8
the array elements are:
3 , 10 , 5 , 16 , 8 , 4 , 2 , 1
```

9- اكتب برنامجاً يصرح عن ثلاث مصفوفات محرفية، الأولى s1(20) ويعين لها القيمة "Happy" والثانية s( ) ويعين لها القيمة "New year" والثالثة s3(40) ويعين لها القيمة لاشيء.

استخدم التوابع المحرفية المناسبة لطباعة ما يلي كل في سطر :

1- قيمة كل من s1 و s2.

2- القيمة الناتجة من إضافة s2 إلى s1 .

3- القيمة الناتجة من إضافة ستة رموز من s1 إلى s3 .

4- القيمة الناتجة من إضافة s1 إلى s3 .

```
# include < iostream.h >
# include < string.h >
main( )
{
    char s1[20] = "Happy";
    char s2[ ] = "New year";
    char s3[40] = "   ";
    cout<<" s1="<< s1<< "s2= "<<s2<< endl;
    cout<<" strcat (s1,s2)= "<< strcat(s1,s2)<< endl;
    cout<<" strncat (s3,s1,6)= "<< strncat(s3,s1,6)<< endl;
    cout<< "<<strcat(s3,s1)= "<<strcat(s3,s1)<< endl;
    return 0;
}
```

10- اكتب برنامجاً يصرح عن سلسلة محرفية ويطبعها كما أدخلت ، ثم يطبعها بعد عكس أحرفها (أي الحرف الأول يصبح الحرف الأخير، والحرف الثاني يصبح الحرف ما قبل الأخير) .

```
# include < iostream.h >
# include < string.h >
main( )
{
    char str[ ]= " a good program ";
    int k;
    cout << str << endl;
    k = strlen (str);
    cout << " k = " << k << endl;
    for (int i = 0; i < k/2; i++)
    {
       char temp = str[i];
       str[i] = str[ k-i-l];
    str[ k-i-1] = temp;
    }
    cout << " string str after reversing its characters is:"<< endl;
    cout << str << endl;
  return 0;
}
```

والخرج هو :

```
a good program
k = 14
string str after reversing its characters is:
margorp doog a
```

الفصل الثامن

التوابع

Functions

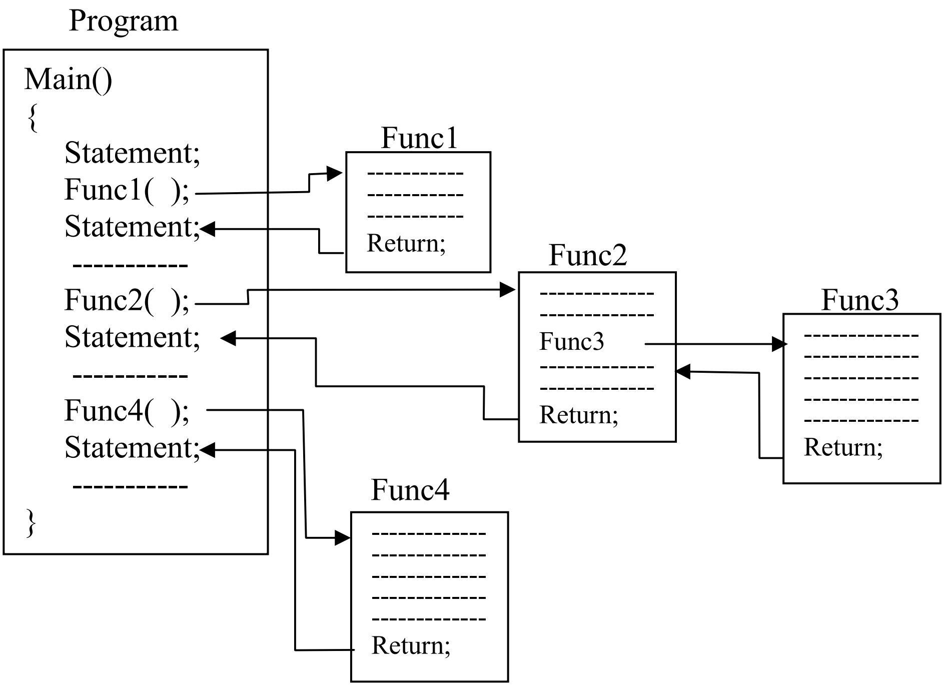
8-1- مقدمة:

معظم البرامج المفيدة التي نعالجها في الحياة العملية هي أكبر بكثير من البرامج التي ذكرناها إلى الآن. يقوم المبرمجون عادة بتقسيم البرامج الكبيرة إلى برامج فرعية (Sub programs) . هذه البرامج الفرعية تدعى توابع (Functions) ويمكن ترجمة واختبار كل منها على حدة، كما يمكن إعادة استخدامها في برامج أخرى مختلفة وحسب الضرورة.

التابع Function هو في الواقع برنامج فرعي مكون من تعليمة أو مجموعة من التعليمات تؤدي مهمة محددة سواء كانت عمليات إدخال، أو إخراج، أو عمليات حسابية، أو منطقية، وله كيان خاص يحتل موقعاً من البرنامج، أي أنه جزء منه.

ولقد ذكرنا أن برنامج ++C يتكون من تابع واحد على الأقل اسمه main() وهذا التابع يستدعى تلقائياً حالما يبدأ تنفيذ البرنامج. ومن ميزات لغة ++C أنها توفر للمبرمج إمكانية كتابة واستخدام التوابع التي يريدها أو يحتاج لها، ويمكنه خزنها في مكتبة خاصة واستعمالها في أي وقت يريد. كما يمكن للتابع main () أن يستدعي توابع أخرى يمكن لبعضها أيضاً استدعاء توابع أخرى أيضاً.

يعطى لكل تابع في البرنامج اسم خاص، وعندما يصادف هذا الاسم يتحول سياق التنفيذ إلى تعليمات التابع المذكور اسمه. وما أن ينتهي تنفيذ التابع حتى يعود التنفيذ إلى السطر التالي أو التعليمة التالية في تعليمات التابع المستدعي. يمكن توضيح هذا الانسياب في التنفيذ كما في الشكل (8-1) التالي:



الشكل (8-1) آلية تنفيذ التوابع

8-2- أهمية استخدام التوابع:

لاستخدام التوابع في البرمجة أهمية كبيرة فهي :

1- تساعد التوابع المخزنة في مكتبة اللغة في اختصار البرنامج ، إذ يكتفى باستعادتها باسمها فقط لتقوم بالعمل المطلوب.

2- تساعد التوابع المخزنة في مكتبة اللغة أو التي يكتبها المبرمج، على تلافي عمليات التكرار في خطوات البرنامج التي تتطلب عملاً مشابهاً لعمل تلك التوابع.

3- تساعد التوابع الجاهزة في تسهيل عملية البرمجة، من حيث الزمن اللازم لذلك والمساحة المطلوبة من الذاكرة.

4- تقسيم البرنامج إلى أجزاء (أي توابع) مستقلة تجعله واضحاً لكل من المبرمج والقارئ والمستخدم على حد سواء.

ولتوضيح أهمية استخدام التوابع في البرمجة بلغة C++ نأخذ المثال التالي:

لو أردنا كتابة خوارزمية لخطوات عمل فنجان من القهوة ، فإننا نكتب مايلي:

1- ضع الماء في إناء غلي القهوة.

2- أشعل النار.

3- سخن الماء حتى يغلي.

4- أضف القهوة إلى الماء الساخن.

5- أضف سكراً إليه.

6- أطفئ النار.

7- صب القهوة في الفنجان.

لو فرضنا أننا نود طلب فنجان قهوة من مقهى قريب، إن خطوات الخوارزمية التي نحتاجها الآن هي خطوة واحدة فقط وهي :

1- استدع فنجاناً من القهوة.

يمكنك الآن أن تتخيل كم توفر من الخطوات باستعمال توابع جاهزة (أو التي يجهزها المبرمج من قبل) بدلاً من خطواتها التفصيلية . وخاصة في برنامج يتطلب عمليات وحسابات كثيرة، وكم يكون البرنامج واضحاً وسهلاً حينئذ.

3-8- أنواع التوابع :

للتوابع نوعان أساسيان:

1- توابع جاهزة : وهي توابع تكون مكتوبة ومعرفة مسبقاً ومخزنة ضمن مكتبة اللغة (standard library) . وتستدعى للتنفيذ بذكر اسمها وتمرير قيم للبارامترات عند الحاجة، ومنها التوابع الرياضية والسلسلية، وسنذكر بعضها لاحقاً.

2- توابع من تصميم المبرمج : وهي توابع يقوم المبرمج بتصميمها ووضع خوارزمية لحلها، ومن ثم يضع التعليمات البرمجية من خلال قواعد وأسس اللغة البرمجية المستخدمة. تُضمَّن هذه التوابع في البرنامج الرئيسي بطريقة معينة، سنتعرف على هذه الأسس والقواعد من خلال لغة البرمجة التي ندرسها وهي ++C.

8-4- مكتبة التوابع القياسية :

مكتبة لغة C القياسية هي مجموعة من التوابع المعرفة مسبقاً وبعض عناصر البرامج التي يمكن الوصول إليها من خلال الملفات الرأسية، وتشكل مكتبة C القياسية جزءاً من بيئة البرمجة الخاصة بلغة ++C.

لقد استخدمنا سابقاً أحد الملفات الرأسية وهو < iostream.h > وذلك عند التعامل مع عمليات الدخل والخرج. أما التوابع الرياضية فمعرفة في الملف الرأسي <math.h> وتسمح للمبرمج القيام بالكثير من العمليات الحسابية الشائعة. ويبين الجدول (8-1) بعض التوابع الرياضية الشائعة ، انظر الملحق E الذي يبين التوابع الموجودة في مكتبة ++C القياسية. والملفات الرأسية المعرفة فيه هذه التوابع.

المثال التالي يوضح استخدام واحدة منها والمعلن عنها في الملف الرأسي < math.h > .

تابع الجذر التربيعي ( )sqrt :

الجذر التربيعي لعدد موجب هو العدد الذي مربعه ذلك العدد. مثلاً الجذر التربيعي للعدد 16 هو 4 لأن مربع 4 هو 16 . يمكن تمثيل التابع بصندوق مغلق، عند إدخال العدد 16 فيه يخرج منه العدد 4. وعند إدخال 4 يخرج منه 2. وهو أشبه بالبرنامج الكامل المكون من دخل – إجراء عمليات على الدخل – خرج . دائماً العمليات التي تحصل في التابع تكون مستترة للمبرمج وما عليه إلاّ أن يعلم أنه لو أدخل قيمة ما سيحصل على قيمة أخرى مربعها هو القيمة المدخلة.

البرنامج التالي يطبع الجذر التربيعي للأعداد من 0 إلى 5.

```
# include < iostream.h >
# include < math.h >
main ()
{
        for (int i=0, i<6; i++)
            cout << " \t << i << " \t" << sqrt (i) << endl;
        return 0;
}
```

لاحظ ضرورة استخدام التوجيه < include < math.h # في السطر الثاني من البرنامج، وذلك لكي يجد المترجم تعريفاً للتابع ()sqrt ، حيث يخبر المترجم بأن التابع قد أعلن عنه في الملف الرأسي < math.h >.

نفذ التابع ( )sqrt باستخدام اسمه كمتغير في تعليمة كالتالي:

```
x = sqrt(y);
```

ويسمى ذلك باستحضار invoking أو استدعاء Calling التابع.

لذلك الأمر sqrt(i) يستدعي التابع sqrt ، المتغير i الذي بين قوسين يسمى بارامير (argument or parameter) ونقول إن متغير البارامير أرسل بالقيمة إلى التابع ولذلك عندما i تساوي 3 ترسل إلى التابع sqrt بالاستدعاء sqrt(3) وتعود منه بالقيمة 1.73205 إلى البرنامج الرئيسي المستدعي.

ملاحظة (1): قيم التابع يمكن أن تستخدم في التعبير الجبري مثل المتغيرات العادية لذلك يمكن أن نكتب:

```
y = sqrt(4);
cout << 2 * sin (x) * cos(x);
```

التعليمة الأولى تعين ناتج استدعاء التابع sqrt من أجل قيمة للبارامير تساوي 4 للمتغير y. أما التعليمة الثانية فتظهر ناتج حساب العلاقة 2*sin(x)*cos(x) على الشاشة.

ملاحظة (2): يمكن تفريع استدعاء التابع كالتالي:

```
y=sqrt(1+2*sqrt(3+4*sqrt(5)));
```

التعليمة السابقة تعين ناتج استدعاء التابع sqrt من أجل قيمة الباراميتر تساوي (1+2*sqrt(3+4*sqrt(5))) للمتغير y. ونحتاج لحساب هذه القيمة إلى استدعاءالتابع sqrt مرتين أولاً من أجل قيمة للباراميتر تساوي 5 وثانياً من أجل القيمة 3+4*sqrt(5)

الجدول (8-1) **بعض التوابع الرياضية في الملف الرأسي** math.h

التابع	**الوصف**	**مثال**
sqrt(x)	الجذر التربيعي للمتحول x	sqrt(900.0) **يعطي** 30.0 sqrt(16.0) **يعطي** 4.0
exp(x)	التابع الأسي e^x	exp(1.0) **يعطي** 2.718282 exp(2.0) **يعطي** 7.389056
log(x)	تابع اللوغاريتم الطبيعي لـ x (ذو القاعدة e)	Log(2.718282) **يعطي** 1.0 Log(7.389056) **يعطي** 2.0
log10(x)	تابع اللوغاريتم العشري لـ x (ذو القاعدة 10)	Log10(1.0) **يعطي** 0.0 Log10(10.0) **يعطي** 1.0 Log10(100.0) **يعطي** 2.0
fabs(x)	تابع القيمة المطلقة لـ x	إذا كانت x>0 عندها fabs(x) هو x إذا كانت x=0 عندها fabs(x) هو 0.0 إذا كانت x<0 عندها fabs(x) هو -x
ceil(x)	تقريب x إلى أصغر عدد صحيح أكبر من x	Ceil(9.2) **يعطي** 10.0 Ceil(-9.8) **يعطي** -9.0
floor(x)	تقريب x إلى أكبر عدد صحيح أصغر من x	Floor(9.2) **يعطي** 9.0 Floor(-9.8) **يعطي** -10.0
pow(x,y)	x مرفوعة للقوة y أي x^y	Pow(2.7) **يعطي** 128.0 Pow(9.2) **يعطي** 81.0
fmod(x,y)	باقي القسمة ذو الفاصلة العائمة لـ $\frac{x}{y}$	Fmod(13.657,2.333 **يعطي** 1.992
sin(x)	تابع الجيب لـ x (x مقدرة بالراديان)	Sin(0.0) **يعطي** 0.0
cos(x)	تابع التجيب لـx (x مقدرة بالراديان)	Cos(0.0) **يعطي** 1.0
tan(x)	تابع الظل لـx(x قيمة مقدرة بالراديان)	Tan(0.0) **يعطي** 0.0

5-8- توابع من تصميم المبرمج:

رغم التنوع الكبير للتوابع في مكتبات لغة C ولغة ++C ، إلاّ أنه مازال غير كاف لتغطية معظم مهام البرمجة . ولذا يحتاج المبرمجون إلى تصميم وتعريف توابع خاصة بهم.

يتألف التابع الذي يقوم المبرمج بتصميمه وبرمجته وتضمينه في برنامجه الرئيسي من ثلاثة عناصر أساسية: الإعلان عن التابع، استدعاء التابع، وتعريف التابع كما هو موضح في الشكل العام (8-2) التالي:

```
# include < iostream.h >
Function prototype(declaration);
main( )
{
        ----------------
        ----------------
        Function calling;
        ----------------
        ----------------
}
        Function definition
        {
              Function – Body;
        }
```

الشكل (8-2) عناصر التابع المصمم من قبل المبرمج

1-5-8- الإعلان عن التابع (Declaration) :

يكون الإعلان عن التابع مفصولاً عن تعريفه. الإعلان عن التابع (declaration) يزود المترجم بكل المعلومات التي يحتاجها ليترجم كامل الملف.

لا يحتاج المترجم لمعرفة كيفية عمل التابع (أي تعليمات التابع)، هو يحتاج فقط لاسم التابع ونوع القيمة المرجعة من التابع، وعدد ونوع البارامترات ، هذه المعلومات موجودة كلها في الإعلان عن التابع.

يسمى إعلان التابع بالنموذج الأولي (proto type) . يتم الإعلان عن التابع قبل استدعائه مثل الإعلان عن المتغيرات ويجب أن يكون دوماً قبل التابع الرئيسي .main( )

الصيغة العامة لتعليمة الإعلان عن التابع هي :

return_type function_name (argument_List);

حيث :

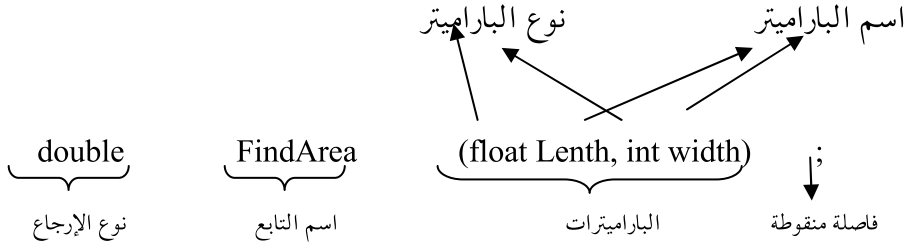
return_type : نوع القيمة المرجعة أي نوع القيمة التي نحصل عليها من تنفيذ التابع المستدعى والتي يمكن أن تكون صحيحة (int) أو حقيقية (float or double) أو محرفية (char) أو لاشيء (void).

function_name : اسم التابع، وهو اسم يختاره المبرمج ويخضع لقواعد إعطاء الأسماء في اللغة. ويفضل أن يكون الاسم معبراً عن عمل التابع.

argument_List : قائمة بأسماء وأنواع البارامترات وتفصل بينها الفاصلة. حيث سيتم إرسال قيم لهذه البارامترات من التابع المستدعي مطابقة بالعدد والترتيب والنوع للقائمة وذلك ليتم تنفيذ التابع المستدعى باستخدامها، تضم القائمة اسم كل بارامتر ونوعه، ويمكن للنوع أن يكون أياً من أنواع البيانات المعرفة والتي مرت بنا سابقاً.

لاحظ انتهاء الإعلان عن التابع بالفاصلة المنقوطة.

الشكل (8-3) التالي يوضح أجزاء تعليمة الإعلان عن التابع FindArea



الشكل (8-3) **أجزاء النموذج الأولي للتابع** FindArea

فيما يلي أمثلة على تعليمات الإعلان عن توابع:

```
int cube (int x);
int sum (int A, int B);
float Area (float y, int x);
```

ملاحظة (1) : يمكن عدم ذكر اسم البارامير في تعليمة الإعلان عن التابع والاكتفاء بذكر النوع فقط. ويجب تكرار كتابة نوع البارامير ولو كانت كلها من النوع نفسه بعدد من المرات مساوٍ لعدد البارامبترات .

فيما يلي أمثلة مقبولة للإعلان عن تابع حساب المساحة area (للتابع قيمة مرجعة حقيقية وبارامبترات حقيقية).

```
float Area (float x, float y);
float Area (float, float);
```

ولكن تعليمة الإعلان التالية :

```
float Area(float x,y);
```

تؤدي إلى حدوث خطأ في أثناء عملية الترجمة، لأنه يجب تحديد نوع كل بارامير من بارامبترات التابع.

بعض التوابع لاتأخذ أية قيمة من التابع المستدعي وبالتالي الصيغة العامة للإعلان عن التابع من دون بارامبترات يأخذ الشكل التالي:

```
return_type    function_name ( );
```

أي نحافظ على الأقواس ( ) بالرغم من عدم وجود أية بارامبترات بداخلها.

فيما يلي أمثلة مقبولة على الإعلان عن تابع:

```
int square (int);
float sum (float, float);
void draw( );
```

8-5-2- استدعاء التابع (Function calling):

لكي يتم تنفيذ التابع يجب تنفيذ تعليمة الاستدعاء، وتؤدي تعليمة الاستدعاء إلى إيقاف تنفيذ البرنامج الرئيسي (التابع المستدعي) والانتقال لتنفيذ التابع المستدعى. ثم العودة بعد الانتهاء منه لمتابعة التنفيذ في البرنامج الرئيسي بدءاً من التعليمة التي تلي تعليمة الاستدعاء، ويكون ذلك عادة باستخدام تعليمة خاصة هي return سنتحدث عنها لاحقاً.

يكون الاستدعاء في أي مكان من البرنامج الرئيسي أو التابع المستدعي نحتاج فيه لتنفيذ التابع الفرعي وذلك حسب الصيغة التالية:

```
function_name ( argument_list );
```

حيث :

function_name : اسم التابع الفرعي وهو الاسم نفسه الذي اختاره المبرمج للتابع في تعليمة الإعلان.

argument_list : قائمة البارامترات : وهي عبارة عن أسماء المتغيرات المعلن عنها في التابع المستدعي والتي سيتم معالجة قيمها في التابع الفرعي المستدعى ، (وهذا يسمى استدعاء باستخدام المرجع Calling by reference) ، أو عبارة عن القيم المراد معالجتها في التابع الفرعي (وهذا يسمى استدعاء بالقيمة Calling by value) ، أو كليهما . ويمكن أيضاً أن يكون الاستدعاء من دون بارامترات إذا كان التابع لايأخذ أية بيانات من التابع المستدعي.

فيما يلي أمثلة لتعليمات استدعاء مقبولة:

```
Square (x);
Sum (z,y);
Draw( );
```

حيث إن البارامترات z,y,x هي متغيرات موجودة في البرنامج الرئيسي المستدعي ولها قيم مخزنة في الذاكرة ومصرح عنها في بداية البرنامج.

8-5-3- تعريف التابع (Function definition):

يتفق تعريف التابع مع النموذج الأولي للتابع ، تماماً من حيث نوع الإرجاع والاسم ولائحة الباراميترات، ويتضمن تعريف التابع أيضاً تعليمات التابع، أي مجموعة التعليمات التي تقوم بالمهمة المطلوبة من التابع، ويتم حصر تعليمات التابع بالقوسين الكبيرين { } . الصيغة العامة لتعريف التابع :

```
return_type   function_name (argument_list)
        {
               function_body;
        }
```

حيث:

return_type : نوع الإرجاع. وهو نوع الإرجاع المحدد نفسه في جزء الإعلان عن التابع.

function_name : اسم التابع الفرعي وهو الاسم نفسه الذي اختاره المبرمج للتابع في جزء الإعلان عنه.

argument_list : قائمة الباراميترات وهي عبارة عن أسماء المتغيرات ونوعها التي تأخذ قيمها من التابع المستدعي لتستخدم في التابع المستدعى. ويمكن أيضاً أن يكون من دون أية باراميترات إذا كان التابع لايأخذ أية بيانات من التابع المستدعي.

ملاحظة (1) : يمكن أن تكون قائمة الباراميترات في جزء تعريف التابع هي نفسها المستخدمة في جزء تعليمة الاستدعاء.

ملاحظة (2) : النموذج الأولي للتابع مطابق تماماً لتعريف التابع وإذا اختلف فسنحصل على خطأ بالترجمة. في الواقع الفرق الوحيد هو وجود الفاصلة المنقوطة في نهاية النموذج الأولي للتابع.

function_body : مجموعة من تعليمات وأوامر لغة C++ التي تشكل جسم التابع. إذا كانت القيمة المرجعة من التابع هي أحد الأنواع التالية char, double, float, int عندئذ يجب أن ينتهي بالتعليمة:

return value;

حيث:

value : هي القيمة التي ستعود إلى التابع المستدعي ونوع هذه القيمة من نوع القيمة المرجعة نفسه. ويمكن أن تكون قيمة ثابتة، اسماً لمتغير، أو تعبيراً سيتم إرجاع ناتج حسابه إلى التابع المستدعي.

ويؤدي تنفيذ التعليمة return إلى العودة إلى البرنامج الرئيسي أو التابع المستدعي والمتابعة بالتنفيذ بدءاً من التعليمة التي تلي تعليمة الاستدعاء.

أما إذا كان التابع من النوع void فلا حاجة عندئذ لهذا الأمر.

مثال (1) :

فيما يلي برنامج يكرر طلب إدخال قيمتين صحيحتين n , m واستدعاء تابع آخر اسمه max وظيفته إيجاد القيمة العظمى بينهما، ومن ثم يطبع القيمة العظمى. يستمر بتكرار ذلك مادامت القيمة m المدخلة لاتساوي الصفر.

```
// return the largest of the two given integers.
# include < iostream.h >
int max (int, int);  // function max prototype
main ()
{
    int m,n;
    do
     {
            cin >> m >> n;
            cout << max (m,n) << endl;
      } while (m!=0);
    return 0;
}
int max (int A, int B)
{
   if (A < B)
      return B;
   else
     return A;
}
```

خرج البرنامج

```
23   43↵
43
0     11↵
11
```

لاحظ إمكانية وجود أكثر من تعليمة return واحدة في التابع، حيث إحدى هذه التعليمات ستؤدي إلى إنهاء تنفيذ التابع والعودة إلى البرنامج الرئيسي ، الأمر return مثل الأمر break . فهو أمر قفز، يقفز خارج التابع الموجود فيه. مع أنه من المعتاد وجود الأمر return في نهاية التابع إلاّ أنه من الممكن وضعه في أي مكان داخل التابع مثل أي أمر آخر. وذلك تبعاً لتحقق الشرط المختبر أم لا.

المتغيرات المدونة في قائمة باراميترات التابع تسمى باراميترات وهمية وهي متغيرات محلية لها وجود فقط في أثناء تنفيذ التابع. تدوين هذه المتغيرات في قائمة الباراميترات يشكل الإعلان عنها. في المثال السابق الباراميترات الوهمية هي : B , A المتغيرات المدونة في استدعاء التابع تسمى باراميترات حقيقية. إنها متغيرات مثل أي متغيرات أخرى في البرنامج الرئيسي يجب أن يعلن عنها قبل استخدامها عند استدعاء التابع. في المثال السابق الباراميترات الحقيقية هي n , m .

في الأمثلة السابقة تم إرسال الباراميترات الحقيقية بقيمها هذا يعني أن قيم تلك الباراميترات خصصت لما يناظرها من الباراميترات الوهمية للتابع. لذلك فإنه في المثال

السابق خصصت قيمة m لـ A وقيمة n خصصت لـ B . عند الإرسال بالقيمة يمكن للبارامترات الحقيقية أن تكون ثوابت أو تعابير جبرية.

على سبيل المثال يمكن استدعاء التابع max كالتالي: max(44,5*m–n);

وهذا سوف يخصص القيمة 44 لـ A وقيمة التعبير الجبري 5*m-n لـ B .

ملاحظة (3) : عند استدعاء التابع يجب تمرير بارامترات حقيقية مطابقة بالعدد والنوع والترتيب للبارامترات الوهمية للتابع.

مثال (2) البرنامج التالي يقوم باستدعاء تابع إيجاد عاملي عدد n حيث إن عاملي عدد موجب !n نحصل عليه بضرب العدد n بكل الأعداد الموجبة والتي هي أقل من n أي:

n! = (n) (n-1) (n-2) … (3) (2) (1)

البرنامج الرئيسي يستخدم حلقة لتحديد الأعداد المطلوب إيجاد العاملي لها وهي من i = - 1 إلى i < 6 . تذكر أن عاملي عدد سالب غير محدد.

```
# include < iostream.h >
int factorial (int);
main ()
{
       for (int i = -1; i<6; i++)
            cout << " factorial ("<< i << " ) = " << factorial (i) << "    ";
       cout << endl;
       return 0;
}
int factorial (int k)
  {
     if (k <= 0)
         return 0;
        int f = 1;
     while (k > 1)
       f * = k--;
    return f;
  }
```

خرج البرنامج هو:

```
factorial (-1) = 0      factorial (0) = 0    factorial (1) = 1
factorial (2) = 2       factorial (3) = 6    factorial (4) = 24
factorial (5) = 120
```

التابع factorial له متغيران محليان f , k والتابع main له متغير محلي وحيد هو i .

8-6- التعاودية (Recursion):

تعاملنا حتى الآن مع مجموعة من التوابع تقوم باستدعاء بعضها بعضاً بأسلوب منهجي وبطريقة هرمية موضحة في الشكل (8-1) . لكن من المفيد في بعض التطبيقات أن تقوم بعض التوابع باستدعاء نفسها. نسمي هذه التوابع التي تستدعي نفسها بالتوابع التعاودية (Recursive function) لأنها تستدعي نفسها بشكل مباشر أو غير مباشر من خلال تابع آخر.

لنناقش مفهوم التعاودية بشيء من التفصيل. حتى يتم استدعاء تابع تعاودي يجب أن يعرف كيفية حل المسألة في أبسط حالتها، نسمي هذه الحالة أو الحالات البسيطة بالحالة أو الحالات الابتدائية (الأساسية) ((base case(s)) . أي عند استدعاء التابع مع حالة ابتدائية فإنه يعيد ببساطة الحل المتعلق بهذه الحالة. أما عند استدعائه مع حالة أكثر تعقيداً فإنه يقوم بتحليل وتقسيم هذه الحالة إلى جزأين بحيث يعرف ماذا يفعل بالجزء الأول وهو الحالة الأساسية (base case) ولكنه لايعرف ماذا يفعل بالجزء الثاني. ولإيقاف التعاودية يجب على الجزء الثاني أن يشابه المسألة الأصلية لكنه يختلف عنها في كونه يتناول حالة أسهل وأبسط من حالة المسألة الأصلية، وعلى اعتبار أن حالة الجزء الثاني تتناول مسألة مشابهة للمسألة الأصلية، لذلك يمكن للتابع أن يعود ليستدعي نفسه لحل مسألة أصغر من المسألة السابقة... وهكذا.

نسمي هذه التعاودية في الاستدعاء (recursive call) بخطوة التعاودية (recursion step) . تتضمن خطوة التعاودية على الكلمة المحجوزة return لأنه يجب

على نتائج هذه الخطوة أن تنضم إلى نتائج الجزء الأول الذي يعرف التابع التعاودي التعامل معه من أجل تكوين حل للمسألة المتناولة وإعادة النتيجة إلى التابع المنادي الذي يمكن أن يكون التابع main.

يتم تنفيذ خطوة التعاودية بينما يظل الاستدعاء الأصلي للتابع مفتوحاً أي لم يتم الانتهاء منه بعد. يمكن للخطوة التعاودية أن تنتج في كل مرة يحتاج فيها التابع التعاودي إلى تقسيم المسألة الجزئية التي يتناولها إلى جزأين. ومن أجل أن نضمن توقف التعاودية يجب أن ننتبه إلى أن التابع لايستدعي نفسه مرة أخرى إلاّ مع مسألة جزئية أصغر وأسهل من المسألة التي نتجت عنها مما يؤدي إلى تقارب المسائل الجزئية الناتجة نحو الحالة أو الحالات الابتدائية التي نعلم حلها. عند الوصول إلى حالة ابتدائية ، يقوم التابع التعاودي بإعادة نتيجة حلها إلى النسخة السابقة عن التابع التي تعيد بدورها نتيجتها إلى النسخة السابقة عنها وهكذا حتى تصل النتيجة إلى الاستدعاء الأصلي الذي يمكن أن يعيد النتيجة إلى التابع المستدعي له وهو مثلاً التابع main.

لنأخذ مثال حساب عاملي عدد من أجل توضيح فكرة التعاودية:

إذا كان لدينا n عدد صحيح غير سالب فإن القيمة !n (n عاملي) تساوي الجداء التالي :

$$(n)\ (n-1)\ (n-2)\ldots\ 1$$

حيث !1 تساوي 1 و !0 تساوي الواحد أيضاً. على سبيل المثال :

$$5! = 5 * 4 * 3 * 2 * 1 = 120$$

يمكن حساب عاملي عدد باستخدام الطريقة التكرارية (iteratively) باستخدام البنية for على الشكل التالي :

```
Fact = 1;
for (int i =1; i < = n, i++)
        Fact * = i;
```

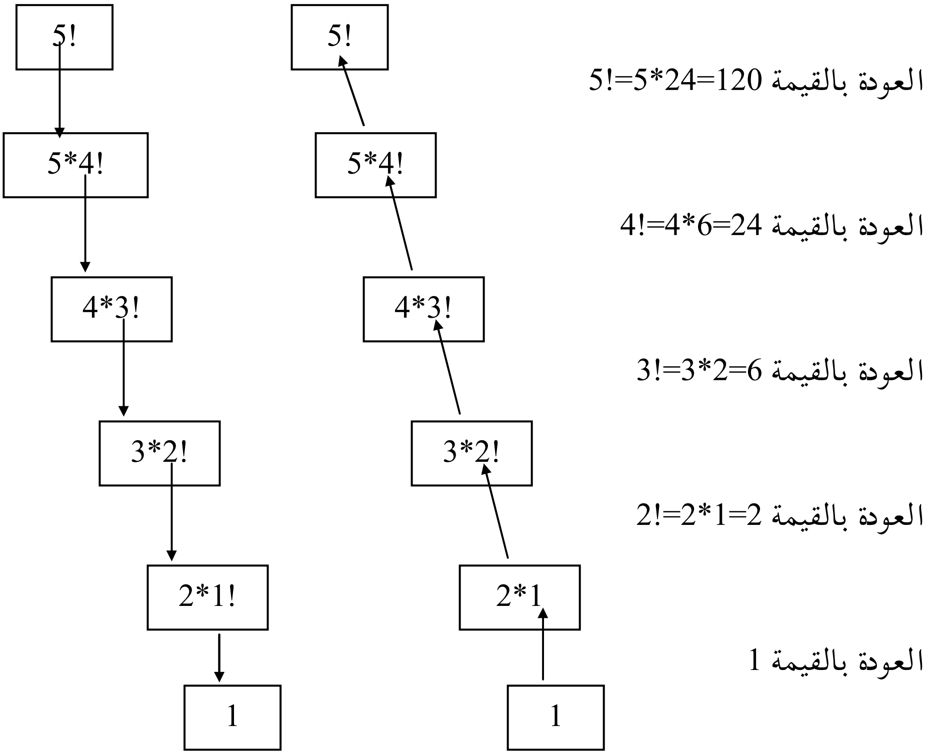
أما بالنسبة للتعريف التعاودي لتابع العاملي فيمكن أن نحصل عليه باستخدام العلاقة التالية :

$$n! = n \cdot (n-1)!$$

فعلى سبيل المثال:

$5! = 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1$
$5! = 5 \cdot (4 \cdot 3 \cdot 2 \cdot 1)$
$5! = 5 \cdot (4!)$

يمكن أن تتم عملية حساب القيمة !5 كما هو مبين في الشكل (4-8) حيث يوضح الجزء (a) منه كيف تتوالى الاستدعاءات التعاودية حتى نصل إلى !1 التي نعرف أنها تساوي القيمة 1 . بعد الانتهاء من التعاودية يوضح الجزء (b) كيف يتم إعادة نتيجة كل استدعاء تعاودي حتى نصل إلى النتيجة النهائية.



(b) القيم المعادة من الاستدعاءات التعاودية (a) معالجة الاستدعاءات التعاودية

الشكل (4-8) **التقييم التعاودي للعدد** !5

مثال (3): البرنامج التالي يستخدم التعاودية من أجل حساب وطباعة قيمة العاملي للأعداد الصحيحة من 0 حتى 10 . يقوم التابع التعاودي بالتحقق أولاً من صحة شرط الاستمرار بالتعاودية أي يتحقق فيما إذا كان العدد number أقل أو يساوي 1 . إذا كان الشرط محققاً (أي أن قيمة number أقل أو تساوي القيمة 1) عندها يعيد fact القيمة واحد ولايوجد حاجة للاستمرار في التعاودية وينتهي البرنامج. أما إذا كانت قيمة العدد number أكبر من القيمة 1 فإن التعليمة :

return number * fact (number – 1);

تسبب إعادة صوغ المسألة لتصبح عبارة عن نتيجة عملية جداء للعدد number مع الاستدعاء التعاودي للتابع fact بقيمة للباراميتر تساوي (number – 1) . لاحظ أن الاستدعاء fact(number-1) سيسعى لحل مسألة أسهل نسبياً من المسألة التي يتناولها الاستدعاء السابق fact (number).

```
// recursive factorial function
# include < iostream.h >
# include < iomanip.h >
unsigned long fact (unsigned long);
main()
{
        for (int i = 0; i < = 10; i++)
            cout << setw(2)<<i<<" ! = " << fact (i) << "\t ";
        cout << endl;
        return 0;
}
// recursive definition of function factorial
unsigned long fact (unsigned long number)
{
        if (number < = 1)
            return 1;
        else
            return number * fact (number -1);
}
```

الخرج هو :

```
0! = 1          1! = 1      2! = 2      3! = 6        4! = 24
5! = 120        6! = 720    7! = 5040   8! = 40320
9! = 362880     10!=3628800
```

8-7- المعرفات ونوع التخزين ومجال الرؤية (Identifiers storage class & scope)

لقد استخدمنا المعرفات (الأسماء) للتعبير عن المتغيرات. وتتصف هذه المتغيرات بعدة خصائص فالتعليمة:

```
int sum = 40;
```

تعطي فكرة عن اسم المتغير sum ونوع بياناته int وحجمها 2 byte وقيمته وهي 40.

وفي هذا الفصل استخدمنا المعرفات للتعبير عن أسماء التوابع المعرفة من قبل المبرمج وسنتعرف في هذه الفقرة على خصائص أخرى وهي نوع التخزين (storage class) ومجال الرؤية (scope).

يحدد نوع التخزين لمتغير الفترة الزمنية لوجوده في الذاكرة. فبعضها يظل موجوداً لفترة قصيرة. بينما يتم إنشاء بعضٍ منها وتدميره عدة مرات. في حين يبقى بعضها الآخر طوال فترة تنفيذ البرنامج.

أما مجال الرؤية لمتغير فتحدد الأماكن التي يمكن منها العودة إليه ضمن نص البرنامج. حيث يمكن العودة لبعضها ضمن كامل البرنامج، أما بعضها الآخر فهو محدود ضمن أجزاء معينة منه.

يوجد في لغة ++C أربعة محددات لنوع التخزين (storage class specifiers) هي: static, register, auto, extern . حيث يساعد محدد نوع التخزين على تحديد نوع تخزين المعرف ومجال رؤيته.

ويمكن تقسيم هذه المحددات الأربعة إلى نوعين هما : نوع التخزين الأوتوماتيكي (automatic storage class) ونوع التخزين الساكن (Static storage class).

تستخدم الكلمتين المحجوزتين auto و register للتصريح عن متغيرات من نوع التخزين الأوتوماتيكي حيث يتم إنشاء هذه المتغيرات عند دخول البرنامج لتنفيذه داخل الكتلة التي وردت فيها التصريحات المتعلقة بها. ويتم بالتالي التعامل معها طالما بقيت هذه الكتلة فعالة، وتدمر بعدها عند تجاوز الكتلة. يمكن للمتغيرات فقط أن تنتمي إلى نوع التخزين الأوتوماتيكي . كذلك الأمر بالنسبة للمتحولات المحلية وباراميترات تابع من التوابع.

المحدد auto يستخدم للتصريح الظاهر عن متحولات ذات نوع تخزين أتوماتيكي.

مثال : التعليمة التالية هي تعليمة تصريح عن متغيرين B,A محليين ينتميان إلى نوع التخزين الأتوماتيكي. أي أنهما موجودان ضمن تعليمات التابع حيث يظهر التصريح عنهما:

```
auto int A , B;
```

تُعد جميع ا لمتغيرات المحلية متغيرات من نوع التخزين الأوتوماتيكي وذلك بشكل افتراضي By default . لذلك لاتستخدم الكلمة المحجوزة auto معها.

تُعد ميزات التخزين الأوتوماتيكي مثالاً على الحد الأدنى من الميزات المطلوب توافرها في البرامج، لأنه لايوجد أي مبرر للإبقاء على المتغيرات في الذاكرة إذا لم تعد هنالك حاجة إليها.

أيضاً الكلمتان static , extern محجوزتان من أجل التصريح عن المعرفات الخاصة بالمتغيرات والتوابع التي تنتمي إلى نوع التخزين الساكن (static storage class) . توجد المتغيرات المعرفة بالطريقة السابقة من النقطة التي يبدأ من عندها تنفيذ البرنامج، ويجري تخصيص موقع تخزين لها في الذاكرة وإعطاؤها قيمة ابتدائية من بداية تنفيذ البرنامج. أما بالنسبة للتوابع فيتم إنشاء أسمائها منذ بداية التنفيذ أيضاً. وعلى الرغم من أن المتغيرات وأسماء التوابع تكون موجودة منذ بداية

تنفيذ البرنامج إلاّ أنه لايمكن استخدامها في أي موضع نريد ضمن نص البرنامج. لذا فنوع التخزين ومجال الرؤية هما موضوعان منفصلان كما سنرى.

المعرف الخارجي extern يخص المتغيرات العامة وأسماء التوابع ، يتم خلق المتغيرات العامة بالتصريح عنها خارج أي تعريف ممكن لتابع. تحافظ المتغيرات العامة على قيمها طوال تنفيذ البرنامج كما يمكن استخدامها واستخدام التوابع ضمن أي تابع يلي نقطة التصريح عن المتغيرات المعنية أو تعريف التابع المراد استخدامه ضمن الملف.

المتغيرات المصرح عنها بوساطة الكلمة المحجوزة static هي متغيرات محلية وتبقى معرفة للتابع الموجودة ضمنه. لكنها على عكس المتغيرات الأوتوماتيكية تحتفظ بقيمتها بعد الانتهاء من تنفيذ التابع . وبالتالي عند استدعاء التابع نفسه مرة أخرى فإن المتحولات المحلية من نمط static ستبقى محتفظة بقيمها نفسها التي وصلت عندها بعد آخر خروج من التابع المذكور.

تقوم تعليمة التصريح التالية بالتصريح عن المتغير count على أنه من النوع static وتعطيه القيمة الابتدائية :

```
static int count = 1;
```

كافة المتغيرات العددية من نوع التخزين الساكن تأخذ القيمة الابتدائية صفر إذا لم يقم المبرمج بتحديد قيمة مغايرة لذلك. وكذلك الأمر بالنسبة لمتغيرات المؤشرات الساكنة التي سوف نتعرض لها في الفصل القادم.

ملاحظة(١): يتحدد مجال الرؤية المتعلق بمعرف ما بجزء البرنامج الذي تستطيع ضمنه التعامل مع هذا المعرف. على سبيل المثال، عندما نصرح عن متغير محلي ضمن كتلة فيمكن التعامل مع هذا المتغير ضمن الكتلة نفسها أو الكتل المتداخلة معها. يوجد أربعة مجالات لرؤية معرف ما هي:

1- مجال الرؤية على مستوى التابع Function scope

2- مجال الرؤية على مستوى الملف File scope

3- مجال الرؤية على مستوى الكتلة Block scope

4- مجال الرؤية على مستوى نماذج التوابع Function _ prototype scope

إذا تم التصريح عن متغير خارج نطاق أي تابع فيكون له مجال للرؤية على مستوى الملف. ويكون مثل هذا ا لمتغير معروفاً من قبل جميع التوابع بدءاً من النقطة التي يظهر فيها التصريح المتعلق به حتى نهاية الملف، وبما أن المتغيرات العامة ونماذج التوابع وتعريفاتها تظهر خارج نطاق أي تابع من التوابع فيكون لها مجال للرؤية على مستوى الملف.

تُعد العناوين Labels (المعرفات المتبوعة بـ " : " والتي تحدد عنوان موقع ضمن نص البرنامج مثل العنوان : Start) هي المعرفات الوحيدة التي لها مجال للرؤية على مستوى التابع. حيث يمكن استخدامها والعودة إليها من أية نقطة ضمن نطاق التابع الذي ورد ذكرها فيه ولايمكن لها أن تستخدم خارج جسم التابع.

تتمتع المعرفات المصرح عنها ضمن كتلة ما بمجال للرؤية على مستوى هذه الكتلة. يبدأ مجال الرؤية من النقطة التي وردت عندها التصريحات عن المعرفات وحتى نهاية الكتلة المحددة بالقوس الختامي { . من بين المعرفات التي لها مجال الرؤية السابق نذكر المتغيرات المحلية المصرح عنها في بداية التابع بالإضافة إلى باراميترات هذا التابع التي تُعد أيضاً متغيرات محلية له.

ملاحظة(٢): عندما تتداخل الكتل ويكون هناك تصريح لمتغيرات في الكتلة الخارجية يحمل اسم المتغيرات نفسه في الكتلة الداخلية، فإنه يتم إخفاء المتغيرات في الكتلة الخارجية حتى نهاية تنفيذ الكتلة الداخلية. وهذا يعني أنه عند تنفيذ الكتلة الداخلية فإنه يرى فقط قيم المتغيرات المحلية الخاصة بتلك الكتلة ولايرى قيم المتغيرات التي تحمل أسماء متغيراتها نفسها والموجودة في الكتل التي تحتويها.

ملاحظة(٣): المتغيرات المحلية المصرح عنها بـ static يبقى مجال الرؤية لها محصوراً ضمن الكتلة التي وردت فيها بالرغم من أنه تم خلقها منذ بداية تنفيذ البرنامج . ولذلك يمكن القول إن دورة حياة المتحول لاتؤثر أبداً في مجال رؤيته.

إن المتغيرات والمعرفات الوحيدة التي لها مجال للرؤية على مستوى نماذج التابع هي البارامترات التي يرد ذكرها في نموذج التابع. كما ذكرنا سابقاً، إن نماذج التوابع لاتتطلب ذكر أسماء البارامترات وإنما يجب تحديد نوع هذه البارامترات فقط. وإذا ما تم ذكر أسماء البارامترات ضمن النموذج فإن المترجم يقوم بتجاهلها وبالتالي يمكن استعمال هذه الأسماء في مكان آخر من البرنامج من دون أن يكون هناك أي مجال للبس أو الغموض.

مثال (4) : البرنامج التالي يبين مجالات الرؤية المختلفة المتعلقة بالمتغيرات العامة والمتغيرات المحلية الأوتوماتيكية والمتغيرات المحلية من نمط static :

```
# include < iostream.h>
void a (void);    // function prototype
void b (void);
void c (void);
int x = 1;     // global variable
main ()
{
     int x = 5;  // local variable to main
     cout << " local x in outer scop of main is:"
            << x << endl;
     {          // start new scope
          int x = 7;
          cout << " local x in inner scope of main is;"
                 << x << endl;
     }          // end new scope
     cout << " local x in outer scope of main is"
          << x << endl;
     a( );     // a has automatic local x
     b( );     // b has static local x
     c( );     // c uses global x
     a( );     // a  reinitialize automatic local x
     b( );     // static local x retains its previous value
     c( );     // global x also retains its value
     cout << " local x in main is: " << x << endl;
     return 0;
}
void a (void)
{
     int x = 25;    // initialized each time a( ) is called
     cout << endl << " local x in a( ) is: " << x
          << " after entering a( ) " << endl;
     ++x;
     cout << " local x in a( ) is: " << x
          << " befor exiting a( ) " << endl;
}
void b (void)
{
     static int x = 50;   // static initialization only
                          // first time b( ) is called.
     cout << endl << " local static x is: " << x
          << " on entering b( ) " << endl;
     ++x;
     cout << " local static x is: " << x
          << " on exiting b( ) " << endl;
}
void c (void)
{
     cout << endl << " global x is: " << x
          << " on entering c( ) " << endl;
     x * = 10;
     cout << " global x is: " << x << " on exiting c( ) " << endl;
}
```

ويكون الخرج كما يلي :

```
local x in outer scope of main is 5
local x in inner scope of main is 7
local x in outer scope of main is: 5

local x in a( ) is:25 after entering a( )
local x in a( ) is:26 before exiting a( )

local static x is:50 on entering b( )
local static x is: 51 on exiting b( )

global x is:1 on entering c( )
global x is:10 on exiting c( )

local x in a( ) is:25 after entering a( )
local x in a( ) is:26 before exiting a( )

local static x is: 51 on entering b( )
local static x is: 52 on exiting b( )

global x is: 10 on entering c( )
global x is: 100 on exiting c( )

local x in main is:5
```

8-8- العملية الأحادية لتحديد مجال الرؤية (Unary scope resolution operator)

في المثال السابق استطعنا أن نصرح عن متغيرات محلية وعامة بالاسم نفسه وتسمح لغة C++ بالوصول إلى المتغير العام الذي يحمل اسم المتغير المحلي نفسه ضمن مجال الرؤية نفسه باستخدام العملية الأحادية لتحديد مجال الرؤية (: :) . ولكن لايمكن استخدام عملية تحديد مجال الرؤية الأحادية للوصول إلى متغير محلي ضمن كتلة خارجية وله اسم متغير محلي نفسه ضمن الكتلة الحالية المعتبرة. نعلم أنه يمكن الوصول إلى

متغير عام مباشرة من دون استخدام العملية الأحادية لتحديد مجال الرؤية ولكن فقط إذا كان اسم ذلك المتغير العام لايشابه اسم أي متغير محلي ضمن مجال الرؤية المعتبرة.

مثال (5) : البرنامج التالي يبين توابع عامة ومتغيرات عامة ومحلية تحمل الاسم نفسه ، كما يوضح إمكانية الوصول إلى كل منها والتعامل معه.

```
# include < iostream.h >
void f( );        // f( ) is global
void g( );      // g( ) is global
int x = 11;    // global x
main ()
{ // begin scope of main
    int x = 22;
        {         // begin scope of inner block
            int x = 33;
            cout << " in block inside main ( ), x = "
                 << x << endl;
        }         // end scope of inner block
    cout << " in main ( ), x =" << x << endl;
    cout << " in main ( ), ::x =" << :: x << endl;
                        // accesses global x
    f( );
    g( );
    return 0;
}
void f( )
{       // begin scope of f( )
        int x = 44;
        cout << " in f( ), x = "<< x << endl;
}       // end scope of f( )
void g( )
{               // begin scope of g( )
        cout << " in g( ), x = " << x << endl;
}       // end scope of g( )
```

والخرج هو :

```
in block inside main ( ), x = 33
in main ( ), x = 22
in main ( ),   :: x = 11
in f( ), x = 44
in g( ), x = 11
```

مثال (6) : البرنامج التالي يبين كيفية استخدام العملية الأحادية لتحديد مجال الرؤية مع المتغيرات المحلية التي لها أسماء المتغيرات العامة نفسها. ومن أجل توضيح الاختلاف للمتغيرين sum المحلي والعام فلقد تم التصريح عن أحدهما على أنه float وعن الآخر على أنه int.

```
// using the unary scope resolution operator
# include < iostream.h >
float sum = 12.45;
main ()
{
        int sum = 8;
        cout << " local value = " << sum << endl
              << " global value = " << :: sum << endl;
        return 0;
}
```

والخرج هو :

```
local value = 8
global value = 12.45
```

8-9- التحميل الزائد للتوابع Function over loading

تسمح لغة C++ باستخدام الاسم نفسه لتوابع مختلفة ولكن بشرط أن يكون عدد ونوع البارامترات مختلفاً فيما بينها. وهذا ما يسمى بزيادة التحميل. نوع الإرجاع (return type) يمكن أن يكون ذاته أو مختلفاً في التوابع الزائدة التحميل مادامت لائحة البارامترات مختلفة ولكن لانستطيع زيادة التحميل للتابع بتغيير نوع الإرجاع فقط.

التعليمات التالية :

```
int  myfunction (int, int);
int  myfunction (long, long);
int  myfunction (long);
```

أدت إلى زيادة تحميل التابع حيث نموذج التابع الأول والثاني مختلفان بنوع البارامترات بينما نموذج التابع الثالث يختلف بعدد البارامترات عن نموذج التابع الأول والثاني.

عند استدعاء تابع مزاد تحميله فإن مترجم لغة C++ يقوم باختيار التابع المناسب بعد أن يتأكد من عدد ونوع وترتيب البارامترات الخاصة بعملية الاستدعاء. نستخدم عادة زيادة التحميل للتابع لخلق عدة توابع لها الاسم نفسه وتقوم بالمهمة نفسها أيضاً ولكن مع أنماط مختلفة للبارامترات.

مثال (7) : البرنامج التالي يستخدم زيادة تحميل التابع max لحساب القيمة العظمى بين عدة قيم :

```
# include < iostream.h >
int max (int, int);
int max (int , int , int);
double max (double , double);
main ()
 {
        cout << max ( 99 , 77 ) << "   "
              << max ( 55 , 66 , 33 ) << "    "
            << max ( 3.4 , 7.2 ) << endl;
      return 0;
 }
 // return the maximum of two given integer values
 int max ( int x , int y)
    {
       return ( x > y ? x : y );
    }
 // return the maximum of three given integer values
 int max ( int x , int y , int z)
   {
         int m = ( x > y ? x : y );
         return ( m > z ? m : z );
   }
// return the maximum of two given real numbers
double max (double x , double y )
 {
         return ( x > y ? x : y );
 }
```

الخرج هو :

```
99    66     7.2
```

تسمح إمكانية زيادة التحميل بإعطاء صفة تعددية الأشكال للعمل ويحدد الشكل حسب نوع البارامیترات وعددها عند الاستدعاء.

عند استخدام زيادة التحميل للتوابع يجب الانتباه إلى عدم التسبب في حالات التباس لايستطيع فيها المترجم الاختيار بدلاً عن المبرمج.

مثال (8) : البرنامج التالي يوقع المترجم في حيرة. إذ أنه لايعرف أيحول المتحول i ذا القيمة واحد (1) والنوع long ضمنياً إلى int أم إلى double ؟ . علماً أن الحالتين كلتيهما ممكنتان. لذا يجب الحذر من الوقوع في مثل هذه الحالات عند استعمال زيادة التحميل.

```
# include < iostream.h >
void display (int);
void display (double);
main ()
{
        long int i=1;
        display (7);
        display(3.5);
        display (i);     // error … compilation is ambiguous!!
        return 0;
}
void display (int i)
{
   cout << " integer " << i << endl;
}
void display (double i)
{
   cout << " double " << i << endl;
}
```

يعطي المترجم التحذير التالي أثناء عملية الترجمة:

```
compiling Ex88.cpp:
Error Ex88.cpp 9: Ambiguity between'display(int)' and display (double)
                        in function main()
warring Ex88.cpp11: 'i'is assigned a value that is never used
                       in function main()
```

8-10- التابع inline :

إن تعريف التابع يعني أن المترجم ينشئ مجموعة واحدة من التعليمات في الذاكرة وينقل التنفيذ إلى هذه المجموعة في كل مرة يتم فيها استدعاء التابع. وإذا كان هنالك حاجة لاستدعاء التابع عدة مرات فإن الذهاب إلى مجموعة تعليمات التابع والعودة منها يقلل من نسبة الأداء للبرنامج وخاصة في التوابع القصيرة جداً ذات تعليمة واحدة أو اثنتين فقط، عادة نستطيع زيادة الفعالية أي السرعة بالتنفيذ عند الإعلان عن التابع بالكلمة المفتاحية inline وعندها سيعمد المترجم إلى نقل كود التابع inline إلى التابع المستدعي مباشرة ولن يكون هناك مراوحة ذهاباً وإياباً وسيصبح التابع المستدعى كما لو كان مكتوباً ضمن التابع المستدعي في موضع الاستدعاء ولكن ذلك سيؤدي إلى ازدياد حجم البرنامج المنفذ لذلك . لاينصح باستخدام مثل هذا التابع inline إلاّ إذا كان التابع المستدعى صغيراً (كما ذكرنا تعليمة أو اثنتين فقط).

مثال (9): البرنامج التالي يدخل عدداً صحيحاً ويستدعي التابع Doublen ليوجد مضاعف العدد عدداً من المرات.

```
# include < iostream.h >
  int Doublen (int);
main()
{
        int x;
        cout << " enter a number to work with : ";
        cin >> x;
        cout << " double " << x << " = ";
        x = Doublen (x);
        cout << x << endl;
        cout << " double " << x << " = " ;
        x = Doublen(x);
        cout << x << endl;
        cout << " double " << x << " = ";
        x = Doublen(x);
        cout << x << endl;
        return 0;
}
int Doublen(int y)
  {
        return 2*y;
  }
```

خرج البرنامج هو :

```
enter a number to work with:3↵
double 3 = 6
double 6 = 12
double 12 = 24
```

يمكن إعادة كتابة المثال السابق باستخدام التابع inline كما يلي:

```
# include < iostream.h >
```

```
inline int Doublen(int y) { return 2*y; }
main ()
{
        int x;
        cout << " enter a number to work with : ";
        cin >> x;
        cout << " double " << x << " = ";
        x = Doublen(x);
        cout << x << endl;
        cout << " double " << x << " = ";
        x = Doublen(x);
        cout << x << endl;
        cout << " double " << x << " = ";
        x = Doublen(x);
        cout << x << endl;
  return 0;
}
```

وخرج البرنامج هو نفسه كالسابق.

8-11- البارامیترات الافتراضیة (Default parameters):

عند استدعاء تابع ما، يجب على التابع المستدعي تمرير قيم لبارامیترات التابع المستدعى والمعلن عنها في النموذج الأولي للتابع وفي تعريفه، هذه القيم مطابقة بالعدد والنوع والترتيب لبارامیترات التابع كما ذكرنا سابقاً. ولكن يمكن للمبرمج أن يحدد بارامیتراً أو أكثر على أنها افتراضية ويعطي قيماً لها . يسرع هذا الأمر في تنفيذ استدعاء التابع لأن هذه البارامیترات لاتحتاج إلى أن تنسخ وبالتالي عند عدم ذكر هذا البارامیتر في أثناء استدعاء التابع فإنه يقوم باستخدام القيمة الافتراضية له .

ملاحظة (1): يجب وضع البارامیترات الافتراضية مزاحة إلى أقصى يمين قائمة البارامیترات عند تعريف التابع. وتعين القيم الافتراضية حسب الترتيب وليس الاسم.

التعليمة التالية صحيحة حيث أعطت البارامتر الثاني قيمة افتراضية:

```
void init (int a, int b = 0);   // 2nd argument = 0 by default
```

والاستدعاءات التالية init (3) و init (2,4) هي أشكال صحيحة لاستدعاء التابع المصرح عنه.

إن المثال التالي خاطئ لأن البارامترات الافتراضية يجب أن تكون متوالية اعتباراً من أقصى اليمين نحو اليسار :

```
void  incorrect ( int a = 3 , int b , int c = 0 );    // error, b has no default value
```

أي إذا بدا النموذج الأولي للتابع كالتالي:

```
long myfunction ( int para1 , int para2, int para3);
```

فإنه يمكن تعيين قيمة افتراضية لـ para2 إذا عينت قيمة افتراضية لـ para3 كذلك يمكن تعيين قيمة افتراضية لـ para1 فقط إذا عينت قيمة افتراضية لكل من Para2 و Para3.

ملاحظة (2): إذا جرى الإعلان عن تابع مع بارامترات افتراضية فلايصح عند تعريف التابع تحديد هذه القيم الافتراضية مرة ثانية.

```
void init (int a = 0);     // declaration of init
void init (int a = 0) { / * function body */} // definition of init error
```

بينما الشكل التالي للإعلان والتعريف بالتابع فهو صحيح :

```
void init (int a = 0);  // declaration of init
void init (int a) { / * function body * / } // ok
```

ملاحظة (3): يسمح بالتعريف الزائد للتابع ذي البارامترات الافتراضية. في هذه الحالة يجب التنبه إلى عدم تجديد القيم الافتراضية مرة أخرى عند إعادة التعريف.

مثال (10): البرنامج التالي يوضح ذلك:

```
# include < iostream.h >
void init (int, int = 3);
void init (int = 7 , int);   // over definition
```

```
main ()
{
        init (2,1);        // display 2,1
        init (4);          // display 4,3
        init ( );          // display 7,3
        return 0;
}
void init (int a , int b)
{
        cout << "a=" << a << "\t" << "b=" << b << endl;
}
```

خرج البرنامج هو :

```
a = 2              b = 1
a = 4              b = 3
a = 7              b = 3
```

عند التعريف الزائد للتابع ( ) init هنا لم يكن يسمح بإعادة تعريف القيمة الافتراضية للبارامیتر الثاني.

ملاحظة (4): يجب ألاّ يذكر اسم البارامیتر الذي يأخذ قيمة افتراضية.

ملاحظة (5): عند استدعاء تابع له عدة بارامیترات افتراضية، فإن حذف أحدهما يؤدي إلى حذف كافة البارامیترات التي تأتي على يمينه ضمن قائمة البارامیترات.

ملاحظة (6): يمكن للقيم الافتراضية أن تكون عبارة عن ثوابت، متحولات عامة، أو استدعاء لتوابع، يمكن أيضاً استخدام البارامیترات الافتراضية مع التوابع .inline

مثال (11): البرنامج التالي يبين استخدام بارامیترات افتراضية في حساب مساحة مكعب.

```
// demonstrates use of default parameter values
# include < iostream.h >
int areacube (int , int = 25 , int = 1);
```

```
main ()
{
        int length = 100;
        int width = 50;
        int hight = 2;
        int area;
        area = areacube (length, width, hight);
        cout << " first time area equals : " << area << endl;
        area = areacube (length, width);
        cout << " second time area equals : " << area << endl;
        area = areacube (length);
        cout << " third time area equals : " << area << endl;
        return 0;
        }
int areacube (int L , int w , int h)
  {
          return (L * w * h);
  }
```

الخرج هو :

```
first time area equals : 10000
second time area equals : 5000
third time area equals : 2500
```

لقد تم في البرنامج السابق إعطاء قيم افتراضية للبارامبتر الثاني والثالث في تعليمة النموذج الأولي لتابع حساب مساحة مكعب. إذا لم تمرر قيمة لـ width فإن المترجم يستعمل width يساوي 25 و hight يساوي واحداً. أما إذا مررنا قيمة لـ width دون تمرير قيمة لـ hight عندها يستعمل قيمة لـ hight مساوية للواحد أيضاً. لكننا لانستطيع تمرير قيمة لـ hight دون تمرير قيمة أيضاً لـ width.

في البرنامج تم إعطاء قيم ابتدائية للأبعاد length و width و hight ثم مررت تلك القيم للتابع areacube لحساب المساحة ومن ثم طباعة تلك القيمة مع عبارة

مناسبة على الشاشة، ثم يعود البرنامج ويستدعى التابع areacube مرة ثانية ويمرر قيماً للبراميتر الأول والثاني ولايمرر قيمة للبراميتر الثالث hight. عندها تستعمل القيمة الافتراضية وهي واحد ثم تحسب المساحة وتعرض قيمتها مع عبارة أخرى مناسبة على الشاشة.

في المرة الثالثة يطلب حساب مساحة المكعب بتمرير قيمة للبراميتر الأول length ومن دون تمرير قيم للبراميتر width و hight وتستعمل عندها القيم الافتراضية لكل منهما في حساب المساحة التي تعرض مرة ثالثة قيمتها مع عبارة أخرى مناسبة على الشاشة.

8-12-تمارين

1- اكتب برنامجاً يدخل n عدداً صحيحاً ثم يحسب ويطبع عدد الأعداد الزوجية even، وعدد الأعداد الفردية odd فيها، استخدم تابعاً فرعياً check يختبر العدد الممرر إليه إن كان زوجياً أو فردياً.

```
# include <iostream.h>
int check (int);
main()
{
    int n, even = 0, odd = 0, d,i,m;
    cout << " enter value of n:";
    cin >> n;
    for ( i = 0; i < n; i++)
       {
          cin >> m;
          d = check (m);
          if( d = = 1)
              even++;
          else
              odd++;
       }
    cout << " \n number of even numbers is = " << even << endl
           << "\n number of odd numbers is = " << odd << endl;
    return 0;
}
int check (int k)
    {
      if (k%2 = = 0)
          return 1;
      else
          return 0;
    }
```

والخرج هو :

```
enter value of n: 10↵
12   1   5   4   3   16   8   23   40    22
number of even numbers is = 6
number of odd numbers is = 4
```

2- اكتب برنامجاً يدخل عددين صحيحين موجبين a , b ، ثم يستدعي تابعاً فرعياً add يقوم بجمع العدد a مع نفسه b مرة ثم يطبع الناتج. استخدم خاصية الاستدعاء المتكرر لنفس التابع الفرعي لجمع a عدداً من المرات مساوياً لـ b.

```
# include <iostream.h>
int add (int, int);
main()
{
    int a,b,c;
    cout << " enter two integer values:";
    cin >> a >> b;
    c = add (a,b);
    cout << " c = " << c << endl;
    return 0;
}
int add (int d, int e)
{
    if (e = = 1)
       return d;
    else
       return d + add (d,e-1);
}
```

والخرج هو :

```
enter two integer values: 12   4↵
c = 48
```

3- اكتب برنامجاً يقبل إدخال عددين صحيحين ويستخدم تابعاً (multiple) ليقرر فيما إذا كان العدد الثاني من مضاعفات العدد الأول. يعود التابع بالقيمة واحد (true) إذا كان العدد الثاني من مضاعفات العدد الأول. وإلاّ فيعود بالقيمة صفر (false). يتوقف البرنامج عن الإدخال والاختبار إذا أدخل عددين متساويين ويساويان الصفر.

```
# include <iostream.h>
int multiple (int, int);
main()
{
          int c,d,e;
    start: cout << " enter two integer values:";
          cin >> c >> d;
          if (( c = = 0) && (d = = 0))
             goto stop;
          e = multiple (c,d);
          if (e = = 1)
             cout << d << " is a multiple of " << c << endl;
          else
             cout << d << " is not a multiple of " << c << endl;
          goto start;
    stop: return 0;
}
int multiple (int a, int b)
{
    if (b% a = = 0)
        return 1;
    else
        return 0;
}
```

والخرج هو:

```
enter two integer values: 2   8↵
8 is a multiple of 2
enter two integer values : 2   7↵
7  is not a multiple of 2
enter two integer values: 0   0↵
```

4-اكتب تابعاً Timesec يأخذ الزمن كثلاثة بارامترات صحيحة (للساعة و للدقيقة و للثانية) ويعود بالزمن مقدراً بالثانية منذ آخر دقة للساعة 12 .

استخدم هذا التابع لحساب وطباعة الزمن مقدراً بالثانية ما بين زمنين محصورين ضمن دورة واحدة قيمتها 12 ساعة.

```
# include < iostream.h >
# include < math.h >
int Timesec (int, int, int);
main()
{
    int h1,m1,s1,h2,m2,s2;
    cout << " enter time1:";
    cin>>h1>>m1>>s1;
    cout << " enter time2:";
    cint >> h2>>m2>>s2;
    if (h1>12)
    h1=h1-12;
    if (h2>12)
    h2=h2-12;
    int c1=Timesec(h1,m1,s1);
    int c2=Timesec(h2,m2,s2);
    int c = fabs(c1-c2);
    cout << "time in seconds:" << c << "seconds" << endl;
    return 0;
}
int Timesec (int h, int m, int s)
{
    return h*60*60+m*60+s;
}
```

والخرج هو :

```
enter time1: 3   14   50⏎
enter time2: 18   20   30⏎
time in seconds: 11140 seconds
```

الفصل التاسع

المراجع والمؤشرات

Pointers and Reference

9-1- المرجع (Reference):

المرجع هو اسم مستعار (Alias) أو اسم آخر لمتغير (هدف Object) يقع في مجال الرؤية الحالي. يلعب المرجع دور الاسم البديل للمتغير وكل ما نجريه على المرجع يطال المتغير. إن الاستعمال الأساسي للمرجع هو مع البارامترات ومع القيم المرجعة من التوابع المكتوبة من قبل المبرمج.

9-1-1- التصريح عن المرجع:

يتم التصريح عن المرجع بكتابة نوع قيمة المتغير، يليه معامل الإسناد & ثم يتبعه اسم المرجع. اسم المرجع هو أي اسم مقبول في لغة C++ . يجب عند التصريح عن المرجع إعطاؤه قيمة ابتدائية (Initialize) هي اسم متغير آخر كما يلي :

```
Data_type & reference_name = variable_name;
```

حيث variable_name هو اسم لمتغير نوعه من نوع اسم المرجع نفسه مثال:

```
int k;
int & b = k;
```

التعليمة الأولى تصرح عن متغير k لمعطيات صحيحة أما التعليمة الثانية فتصرح عن b مرجع لقيمة صحيحة هي k.

مثال (1) : البرنامج التالي ينشأ مرجعاً من دون إعطائه قيمة ابتدائية وهذا خطأ سيعلمنا به المترجم.

```
// references must be initialized
# include < iostream.h >
```

```
main ()
{
        int a = 4, & y;     // Error : y must be initialized
        cout << " a = " << a << "\t" << "y=" <<y<< endl;
        y = 7;
        cout << "a ="<< a << "\t" << "y =" << y << endl;
        return 0;
}
```

الخرج هو :

```
Compibing Ex91.CPP
Error Ex91.CPP 5: Reference variable 'y' must be initialized
                              in function main ( )
```

مثال (2) : البرنامج التالي ينشئ مرجعاً ويعين له قيمة ابتدائية.

```
# include < iostream.h >
main ( )
{
        int a=4, & y=a;
        cout << "a =" << a << "\t" << "y =" << y << endl;
        y = 7;
        cout << "a =" << a << "\t" << "y =" << y << endl;
        return 0;
}
```

والخرج هو :

```
a = 4                    y = 4
a = 7                    y = 7
```

لنلاحظ التعليمات التالية :

```
int k=5;          // normal definition
int & b=k;        // k and b now refer to the same int
int m=b;          // m = b = k = 5
b = 12;           // k is 12 as well
```

التعليمة في السطر الثالث تصرح عن متغير صحيح مساوٍ لـ b. وبما أن b مرجع لقيمة صحيحة k وتساوي 5. إذن m تأخذ القيمة 5 نفسها . أما التعليمة الأخيرة فتعين لمرجع القيمة الصحيحة b (والتي هي أيضاً k) القيمة 12.

ملاحظة (1): لايمكن إعادة تمهيد قيمة جديدة لمرجع. أي إعادة إعطائه قيمة ابتدائية ثانية مختلفة. والقيمة التي يبدأ بها تبقى دائمة له. أي أن المرجع لقيمة صحيحة أولى لايصبح مرجعاً لقيمة صحيحة أخرى. التعليمات التالية توضح ذلك:

```
int k;
int & b=k;      // k and b now refer to the same int
b=5;            // k = 5
b++;            // k is incremented to 6
```

التعليمة الأخيرة تعني أن القيمة الصحيحة التي مرجعها b يجب زيادتها بواحد.

ملاحظة (2): نستطيع أن نغير القيمة k عن طريق مرجعها b ، تماماً كما نغيرها عن طريق اسمها k.

ملاحظة (3): يمكن تقييد هذه العملية وعدم السماح بتغيير القيمة عن طريق مرجعها وذلك بالتصريح عن مرجع لقيمة ثابتة مثال:

```
int k=5;        // k is variable
const int & b=k;
k=10;       // ok k is variable
b = 12;         // error, b is constant
```

أي أن b عينت لها القيمة الأولية لـ k المساوية لـ 5 وستبقى كذلك.

k متغير صحيح مسموح لقيمته أن تتغير وتأخذ القيمة 10 بينما b مرجع لقيمة صحيحة ثابتة تساوي 5 ولايمكن لها أن تصبح 12.

9-1-2- تمرير بارامترات التوابع بوساطة المراجع:

تعلمنا أن التوابع مقيدة بأمرين : البارامترات التي تمرر بقيمها by value وتعليمة الإرجاع التي لاتستطيع إلاّ إرجاع قيمة واحدة فقط. عندما ترسل البارامترات

بالقيمة. هذا يعني أن التعبير المستخدم في استدعاء التابع يقدر أولاً ثم تخصص قيمته الناتجة للبارامير المقابل في قائمة بارامترات التابع قبل بدء تنفيذ التابع.

على سبيل المثال في استدعاء التابع cube(x) إذا كانت قيمة x هي 4 فإن القيمة 4 تنقل إلى المتغير المحلي n قبل بدء تنفيذ التابع. حيث إن القيمة 4 تستخدم فقط داخل التابع ، والمتغير x لايتأثر ، ونستطيع القول إن المتغير x هو بارامير قابل للقراءة فقط (read-only).

طريقة إرسال البارامترات تسمح باستخدام التعابير العامة بدلاً من البارامير الحقيقي في استدعاء التابع. على سبيل المثال التابع cube يمكن أيضاً أن يستدعى بـ cube(3) ، أو cube(2*x-3) ، أو cube(2*sqrt(x)-cube(3)).

في كل حالة يقدر التعبير الذي بين قوسين بقيمة وحيدة ترسل إلى التابع.

طريقة الاستدعاء بالقيمة عادة هي المطلوبة للتوابع. فهي تجعل التابع أكثر استقلالاً ومحفوظاً من أخطاء غير مقصودة. على أية حال تحتاج التوابع أحياناً إلى أن تغير قيمة البارامير المنقول لها. هذا يمكن عمله بوساطة التمرير بالمرجع.

لتمرير البارامير بالمرجع بدلاً من التمرير بالقيمة ، يتم ببساطة بإلحاق المؤثر & بنوع البارامير في قائمة البارامترات للتابع. هذا يجعل المتغير المحلي مؤشراً للبارامير الحقيقي المرر إليه. لذلك فإن البارامير الحقيقي قابل للقراءة والكتابة بدلاً من القراءة فقط. عندئذ أي تغيير للمتغير المحلي داخل التابع سوف يسبب التغيير نفسه للبارامير الحقيقيي الذي مرر إليه.

لاحظ أن البارامترات المنقولة أو الممررة بالقيمة ، تسمى بارامترات ذات قيمة (value parameters) والبارامترات الممررة بالمرجع تسمى بارامترات مرجعية (reference parameters).

مثال (3) : البرنامج التالي يدخل عددين صحيحين ثم يمررهما إلى التابع swap الذي يقوم بتبديل قيمهما، تمرر البارامترات إلى التابع بالقيمة.

```
# include < iostream.h >
void swap (int , int );
main ()
{
        int n, m;
        cout << "enter two integer values :";
        cin >> n >> m;
        cout <<"n and m in main() before calling swap"<<endl;
        cout << "n=" << n << "\t" << "m=" <<m<< endl;
        swap (n,m);
        cout<<"n and m in main() after calling swap"<<endl;
        cout <<" n="<< n << "\t" << "m="<<m<<endl;
        return 0;
}
void swap (int x, int y)
{
        cout << " x and y in swap before swaping"<<endl;
        cout << "x="<< x << "\t"<<"y="<<y<< endl;
        int temp = x;
        x = y;
        y = temp;
        cout <<"x and y in swap after swaping"<<endl;
        cout << " x = " << x << "\t" << " y = " << y << endl;
}
```

الخرج هو :

```
enter two integer values: 12  7↵
n and m in main() before calling swap
n = 12           m = 7
x and y in swap before swaping
x = 12           y = 7
x and y in swap after swaping
x = 7            y = 12
```

```
n and m in main() after calling swap
n = 12          m = 7
```

نلاحظ من الخرج أن التابع swap ، بادل قيمتي x و y ولكن قيمتي n , m بعد الانتهاء من استدعاء التابع ظلت كما هي لم تتغير. لماذا ؟ لأن البارامترات مررت للتابع بالقيمة. في مثل هذه الحالة ينشئ التابع تلقائياً متغيرات جديدة تدعى x و y لتخزين القيم الممررة إليه. ثم يعمل على تلك المتغيرات الجديدة. لذا عند انتهاء تنفيذ التابع ورغم أنه قام فعلياً بتغيير x إلى 7 و y إلى 12 ، فإن قيمة المتغيرين n و m في البرنامج الذي استدعى التابع بقيت كما هي ولم تتغير.

لنعيد كتابة البرنامج السابق ولكن بتمرير البارامترات بالمرجعية بحيث يستطيع التابع الوصول إلى المتغيرات الأصلية n و m بدلاً من إنشاء متغيرات جديدة، والتعامل معها.

لكي نحقق ذلك نقوم بإضافة المؤثر & إلى نوع بيانات البارامير في التعريف والتصريح عن التابع.

مثال (4) : البرنامج التالي يعمل على تبديل قيم متغيرين مررت قيمهما إلى التابع swap بالمرجعية:

```
# include < iostream.h >
void swap (int & , int & );
main ()
{
        int n, m;
        cout << "enter two integer values :";
        cin >> n >> m;
        cout << "n=" << n << "\t" << "m=" <<m<< endl;
        swap (n,m);
        cout << "n=" << n << "\t" << "m=" <<m<< endl;
        return 0;
}
```

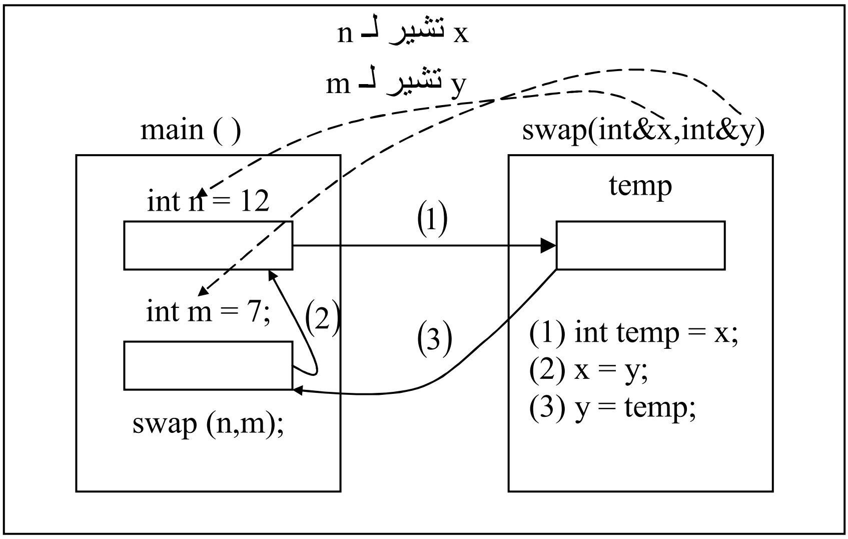
```
void swap (int & x, int & y)
{
        cout << "x="<< x << "\t"<<"y="<<y<< endl;
        int temp = x;
        x = y;
        y = temp;
        cout << " x = " << x << "\t" << " y = " << y << endl;
}
```

والخرج هو :

```
enter two integer values: 12  7↵
n = 12          m = 7
x = 12          y = 7
x = 7           y = 12
n = 7           m = 12
```

المؤثر & يلي int في التصريح والتعريف للتابع. يُبلغ هذا المؤثر المترجم بأن يمرر هذه البارامترات بالمرجعية، أي أن البارامتر x هو مرجع إلى n و y هو مرجع إلى m (لاحظ عدم استعمال & في استدعاء التابع).

الشكل (9-1) يبين كيف يعمل التابع swap على المتغيرين n , m في main ( ) حيث إن التابع swap ليس له متغيرات x , y خاصة به كونها مجرد أسماء مستعارة لـ m,n.



الشكل (9-1) التمرير بالمرجع

نلاحظ أن المرجع هو اسم آخر للمتغير نفسه. فالاسم x يشير إلى الموقع نفسه في الذاكرة الذي يشير إليه n . كما أن y يشير إلى الموقع نفسه الذي يشير إليه m.

الجدول (9-1) يلخص الفرق بين التمرير بالقيمة والتمرير بالمرجعية

الجدول (9-1) التمرير بالقيمة مقابل التمرير بالمرجعية

التمرير بالقيمة	**التمرير بالمرجعية**
- int x;	- Int & x;
- البارامیتر الأساسي x هو متغیر محلي	- البارامیتر الأساسيx هو مؤشر محلي
- یكون مطابق للبارامیتر الحقیقي	- یكون مرادف للبارامیتر الحقیقي
- لایمكن تغییر البارامیتر الحقیقي	- یمكن تغییر البارامیتر الحقیقي
- البارامیتر الحقیقي یمكن أن یكون ثابتاً أو متغیراً أو تعبیراً جبریاً	- البارامیتر الحقیقي یجب أن یكون متغیراً.
- البارامیتر الحقیقي یقرأ فقط	- البارامیتر الحقیقي یقرأ ویكتب.

مثال (5) : البرنامج التالي يبين الفرق بين تمرير البارامیتر بالقيمة والتمرير بالمرجعية.

```
# include < iostream.h >
void g (int, int & );
main ()
{
        int a = 11 , b = 13;
        cout << "a=" << a << "\t" << "b=" <<b<< endl;
        g(a,b);
        cout << "a=" << a << "\t" << "b=" <<b<< endl;
        return 0;
}
void g(int x, int & y)
{
        x = 33;
        y = 20;
}
```

والخرج هو :

```
a = 11          b = 13
a = 11          b = 20
```

عند استدعاء التابع g(a,b) أرسلت a بقيمة إلى x. وأرسل b بمرجع إلى y. لذلك x هو متغير محلي خصصت له قيمة a وهي 11 بينما y هي اسم مستعار للمتغير b الذي قيمته 13. التابع g(a,b) يعين القيمة 33 للمتغير x لكن ليس لها تأثير في a. بينما تعيين القيمة 20 للمتغير y يعني تعيين القيمة نفسها أي 20 للمتغير b. لذلك عند انتهاء تنفيذ التابع فإن a تبقى لها القيمة الأصلية 11 بينما b لها القيمة الجديدة 20. البارامتر الحقيقي a يقرأ فقط بينما البارامتر الحقيقي b يقرأ ويكتب.

ملاحظة (4): نحتاج إلى استخدام البارامتر المرجعي عندما يرجع التابع أكثر من قيمة حيث يمكن للتابع أن يرجع قيمة وحيدة باستخدام التعليمة return. لذلك في حالة إرجاع أكثر من قيمة فإن البارامتر بمرجع يمكن أن يقوم بهذا العمل.

مثال (6) : البرنامج التالي يدخل نصف قطر دائرة ويستدعي تابعاً فرعياً circle لحساب مساحة ومحيط تلك الدائرة ثم يطبع الناتج.

```
# include < iostream.h >
void circle (float &, float & , float );
main ()
{
        float r, a, c;
        cout<<"enter radius:";
        cin >> r;
        circle (a,c,r);
        cout << "area=" << a << "\t" << "circumference=" <<c<< endl;
        return 0;
}
void circle(float & area, float & circumference,float r)
{
        const double pi = 3.14159;
        area = pi * r * r;
        circumference = 2 * pi * r;
}
```

من أجل التصريح عن المتغير circumference و area على أنها متغيرات حقيقية مرجعية يجب استخدام طريقة الكتابة التالية :

```
float & area , & circumference
```

بدلاً من الطريقة الخاطئة :

```
float& area, circumference
```

أو الطريقة الخاطئة:

```
float  &area, circumference
```

9-1-3- تمرير البارامیتر بمرجع ثابت:

نلجأ عادة إلى تمرير البارامیتر بمرجع لسببين:

1- إذا كان على التابع تغيير قيمة البارامیتر الحقيقي كما تم في التابع swap. عند ذلك يجب تمرير البارامیتر بمرجع.

2- إذا كان البارامیتر الحقيقي الممرر للتابع يشغل حيزاً كبيراً من الذاكرة في التخزين (كصورة ما). عندئذ فإنه من الكفاءة التمرير بمرجع لمنع مضاعفة الحيز الذي نحتاجه للتخزين ولكن على أية حال فإن ذلك يسمح للتابع بتغيير قيمة البارامیتر.

إذا كنا لانرغب أن يقوم التابع بتغيير قيمة البارامیتر وإنما فقط طباعته مثلاً عندئذ تمرير البارامیتر بمرجع يشكل خطورة ولغة c++ أعطت بديلاً وهو الانتقال بمرجع ثابت (constant reference) . إنه يعمل بطريقة الانتقال نفسها بمرجع ماعدا أن التابع ممنوع من تغيير قيمة البارامیتر. والتأثير هو أن التابع يستطيع الوصول إلى البارامیتر الحقيقي بوساطة البارامیتر الوهمي في التابع. لكن قيمة البارامیتر الوهمي لايمكن أن تتغير في أثناء تنفيذ التابع. والبارامیتر المنقول عندها يسمى قراءة فقط لأنه لايمكن أن نغير (نكتب) محتويات هذا البارامیتر .

مثال (7) : البرنامج التالي يوضح الطرق الثلاث لتمرير البارامیتر إلى التابع:

```
# include < iostream.h >
```

```
void g (int, int &, const int& );
main ()
{
        int a = 10 , b = 20 , c = 30;
        cout << "a=" << a << "\t" << "b=" <<b<<"\t"<<"c="<<c<<endl;
        g(a,b,c);
        cout << "a=" << a << "\t" << "b=" <<b<< "\t"<<"c="<<c<< endl;
        return 0;
}
void g(int d, int & e, const int & f)
{
        d+ = f;
        e+ = f;
        cout <<"d="<<d<<"\t"<<"e="<<e<<"\t"<<"f="<<f<<endl;
}
```

والخرج هو :

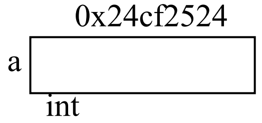
```
a = 10          b = 20          c=30
d = 40          e = 50          f=30
a = 10          b = 50          c = 30
```

لقد غير التابع الباراميترات الوهمية e,d ولكنه لم يستطع أن يغير f. تغيير التابع للباراميتر d لم يؤثر في الباراميتر الحقيقي a لأنه أرسل بقيمة. تغيير التابع للباراميتر e له التأثير نفسه في الباراميتر الحقيقي b لأنه أرسل بمرجع. إرسال الباراميترات بمرجع ثابت يستخدم غالباً مع التوابع التي تتعامل مع أهداف كبيرة مثل المصفوفات، أما أهداف الأنواع الأساسية (الأعداد الصحيحة والأعداد الحقيقية ...الخ) فيتم إرسالها عادة إما بالقيمة (إذا كنا لانرغب في تغيير التابع لها) أو بمرجع (إذا كنا نرغب في تغيير التابع لها).

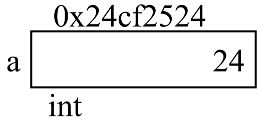
لقد ذكرنا سابقاً أنه عند الإعلان (التصريح) عن أحد المتغيرات فإننا نحدد نوعه واسمه وعنوانه بالذاكرة. التصريح التالي :

```
int a;
```

يحدد الاسم a لنوع القيمة الصحيحة في عنوان ما بالذاكرة ولنفرض أن العنوان هو ox24cf2524 في النظام الست عشري وهذا مشار له بـ 0x قبل الرقم عندها يمكن تمثيل a كالتالي:



حيث يمثل الصندوق مكان تخزين المتغير في الذاكرة . اسمه a على اليسار، نوعه int في الأسفل، عنوانه ox24cf2524 في الأعلى. ولنفرض أن قيمة المتغير معروفة فتمثل بقيمتها داخل الصندوق:



يمكن التعامل مع قيمة المتغير بوساطة اسمه. فمثلاً يمكن طباعة قيمة المتغير a بالأمر التالي :

```
cout << a;
```

أما عنوان المتغير فيمكن التعامل معه بمعامل العنوان address operator . فمثلاً يمكن طباعة عنوان المتغير a بالأمر التالي:

```
cout << & a;
```

معامل العنوان & يعمل على اسم المتغير لينتج العنوان، إن له أسبقية تنفيذ مثل معامل النفي المنطقي Not وعامل الزيادة المسبقة ++.

لقد عرفنا المرجع بأنه اسم مرادف للمتغير ويعلن عنه باستخدام المعامل المرجعي & والذي يلحق بنوع المرجع.

مثال (8) : البرنامج التالي يوضح العلاقة بين اسم المتغير ومرجعه وكذلك قيمة عنوان كل من المتغير ومرجعه:

```
# include < iostream.h >
```

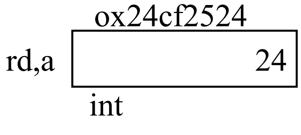
```
main ()
{
    int a , b;
    int &rd=a;       //  rd is reference of a
    cout<<"enter two integer values:";
    cin >> a >> b;
    cout << "a=" << a << " , " << "rd = " << rd << endl;
    a++;
    cout << " a = " << a << " , " << "rd = " << rd << endl;
    rd* = 2;
    cout << " a =" << a << " , " << "rd="<< rd << endl;
    cout << "&a=" << &a<<" , " << "&rd=" << &rd<<endl;
    cout << "&b = " << &b << endl;
    return 0;
}
```

والخرج هو :

```
enter two integer values: 3  5↵
a=3        ,      rd=3
a=4        ,      rd=4
a=8        ,      rd=8
&a = ox24cf2524        ,       &rd = ox24cf2524
&b = ox24cf2526
```

الاسمان a , rd هما اسمان مختلفان للمتغير نفسه ولهما العنوان نفسه لأنهما للموقع نفسه في الذاكرة . زيادة قيمة a أدى إلى زيادة قيمة rd وجداء rd بـ 2 أدى كذلك لجداء a بـ 2 .

التمثيل التالي يوضح عمل المرجع :



لقد رأينا أن المعامل & له استخدامات عديدة في لغة c++ . عندما يتقدم اسم المتغير فإنه يعيد العنوان لهذا المتغير. عند استعماله بعد النوع في الإعلان عن المتغير فإنه يعلن هذا المتغير على أنه مرادف للمتغير الذي حدد اسمه في تعليمة الإعلان بعد =.

أما عند استعماله بعد النوع في الإعلان عن معاملات التابع فإنه يعلن المعامل ليكون معاملاً مرجعياً للمتغير المرسل إليه. كل هذه الاستخدامات تمثل اختلافات للنمط نفسه. إن علامة & تشير إلى العنوان الذي تخزن فيه القيمة.

إن تخزين أي برنامج في الذاكرة يشغل حيزاً معيناً من العناوين. وهذا يعني أن كل متغير وكل تابع في البرنامج يبدأ عند عنوان ما. وتعتمد العناوين الفعلية التي تحتلها متغيرات البرنامج على عدة أمور، كالجزء من الذاكرة الذي يشتغل فيه البرنامج. وحجم نظام التشغيل، وما إذا كان هناك برامج أخرى مشتغلة في الذاكرة أم لا . لهذه الأسباب قد لاتحصل على العناوين نفسها التي حصلت عليها عند تشغيل هذا البرنامج .

9-2- المؤشرات (pointers):

المؤشر هو متغير يحمل عنوان موقع في الذاكرة. نحن نعلم أن المتغير له قيمة ما مخزنة في موقع ما في الذاكرة. متغير العدد الصحيح int يحمل عدداً ، أما متغير من نوع char فيحمل حرفاً. المؤشر هو متغير يحمل عنواناً في الذاكرة.

نحن نعلم أن ذاكرة الحاسوب مقسمة إلى مواقع تخزين مرقمة متتابعة . لكل من هذه المواقع عنوان في الذاكرة. وكل متغير أياً كان نوعه يتوضع في موقع واحد في عنوان معين من الذاكرة. لايحتاج المبرمجون عادة لمعرفة العنوان الخاص بمتغير معين لأن المترجم يتكفل بهذه التفاصيل. لكن إذا احتجنا لهذه المعلومات، علينا استخدام المعامل & قبل اسم المتغير. راجع المثال (8) السابق . كذلك فإن المترجم يعرف كمية الذاكرة التي يحتاجها لتخزين كل متغير من خلال تعليمة التصريح التي تعلمه عن نوع المتغير. لذلك إذا صرحت عن متغير أنه من النوع unsigned long ، فإن المترجم سيعرف أن عليه حجز 4 بايت لأن كل unsigned long يحتاج إلى 4 بايت.

9-2-1- التصريح عن المؤشرات وإعطاؤها قيماً ابتدائية

يعلن عن المؤشر بالشكل :

```
data-type*pointer-name;
```

حيث إن :

data-type نوع القيمة التي يشير لها المؤشر.

pointer-name اسم المؤشر

* معامل عندما يستخدم بالشكل السابق فإنه يصرح أن ما بعده متغير من نوع مؤشر يشير إلى قيمة من النوع الذي قبله.

عندما نعلن عن متغير صحيح من نوع int فإنه يعد ليستوعب عدداً صحيحاً. وعندما نعلن عن متغير مؤشر فإنه يعد ليستوعب عنواناً. إذاً المؤشر هو نوع متغير خاص أعد لكي يحفظ عنواناً لكائن ما في الذاكرة مثال:

```
int*x,y;
```

تعليمة تصريح تعلن أن x مؤشر لعدد صحيح و y متغير صحيح. ولكي يكون كل من x و y مؤشراً لعدد صحيح يجب أن يعلن عنهما بالشكل التالي:

```
int*x , *y;
```

ملاحظة (5): يجب إعطاء المؤشرات قيماً ابتدائية إما عند التصريح عنها أو باستخدام تعليمات التعيين. يمكن إعطاء القيمة 0 أو Null أو أي عنوان آخر كقيمة ابتدائية لمؤشر، حيث يؤشر المؤشر الحاوي على 0 أو Null على لاشيء.

```
int x=4;
int * y;
y = & x;
```

التعليمة ;y = &x تقوم بتعيين عنوان المتغير x إلى المؤشر y. نقول بعدها إن المتغير y يؤشر على x .

ملاحظة (6): إن التصريح عن المتغيرات يجب أن يسبق تعليمة تعيين عناوين هذه المتغيرات في تعليمة التصريح عن المؤشرات. كما يجب التأكد من تطابق أنواع بيانات المؤشرات وبيانات المتغيرات. فالتصريح التالي يُعد غير

صحيح لكون المتغير من النوع int (صحيح). والمؤشر من النوع double (حقيقي مضاعف الدقة).

```
int n;
double* nptr = & n;
```

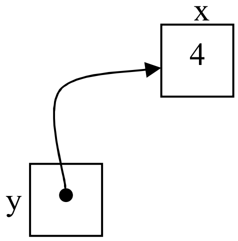
كما لايمكن تعيين العنوان للمؤشر في تعليمة التعيين. لذلك التعليمة التالية غير صحيحة.

```
int* nptr = 0023FF00;
```

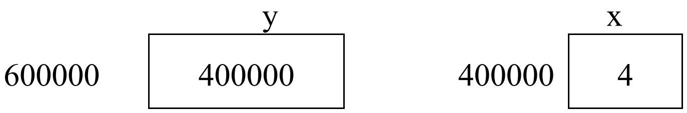
ويستثنى من هذا القيد كما ذكرنا القيمة 0 والتي تسمى Null pointer . إذ تُعد تعليمات التعيين التالية صحيحة.

```
int* nptr = 0;
int* nptr = NULL;
```

يبين الشكل التالي تمثيلاً بيانياً للذاكرة بعد تنفيذ عملية التعيين السابقة. نلاحظ في الشكل أنه تم تمثيل علاقة التأشير بوساطة سهم من المؤشر إلى الغرض المؤشر عليه.



أما الشكل التالي فيمثل تمثيلاً آخر لعملية التأشير بفرض أن عنوان المتغير x في الذاكرة هو 400000 وأن عنوان المؤشر y هو 600000 .



مثال (9) : البرنامج التالي يعين p مؤشراً لقيمة صحيحة n ثم يطبع تلك القيمة عن طريق اسم مؤشرها.

```
# include < iostream.h >
main ()
{
        int n = 40;
        int* p=&n;       // p points to n
        cout << " * p = " << *p << endl;
        return 0;
}
```

والخرج يكون :

```
*p = 40
```

من المثال السابق نلاحظ أن p* هي مرادف لـ n.

عامل العنوان & وعامل إعادة المرجعية * (Dereferencing) هما عكس بعضهما.

يمكننا أن نكتب أن :

n = = *p

عندما :

p = = &*p

ويمكننا التعبير عن ذلك أيضاً بـ n = = *&n

وأيضاً p = = & n

مثال (10) : البرنامج التالي يحدد مؤشراً p لقيمة صحيحة n ومرجعاً r للقيمة نفسها.

```
# include < iostream.h >
main ()
{
        int n = 40;
        int* p=&n;       // p points to n
        int& r=*p;       // r is a reference for n
        cout << " r = " << r << endl;
        return 0;
}
```

والخرج هو :

```
r = 40
```

حيث p مؤشر صحيح لـ n و r مرجع للقيمة التي يشير إليها p وبالتالي تتحدد قيمة r بالقيمة التي يشير إليها p وهي قيمة n.

9-2-2- عامل التعيين غير المباشر (The indirection operator):

رأينا في المثال (9) كيف تسمح المؤشرات بالوصول غير المباشر لقيم المتغيرات التي تدل عليها هذه المؤشرات.

فلو كان لدينا تعليمتا التصريح التاليتان:

```
int n;
int* nptr=&n;
```

ولوضع قيمة في المتغير n ولتكن 20 فإنه يتم ذلك بإحدى الطريقتين التاليتين:

1- بطريقة مباشرة عن طريق تعليمة التعيين:

```
n =20;
```

2- بطريقة غير مباشرة باستخدام عامل التعيين غير المباشر والذي يرمز له بنجمة (*) .

حيث يمكن استخدام هذا العامل لوضع القيمة 20 في المتغير n على النحو التالي:

```
* nptr = 20;
```

9-2-3- تمرير بارامیترات التوابع بوساطة المؤشرات:

إن تمرير البارامیتر بالمؤشر بدلاً من التمرير بالقيمة يتم بإلحاق المؤثر (*) بنوع البارامیتر في قائمة البارامیترات للتابع. وهذا يجعل المتغير المحلي مؤشراً للبارامیتر الحقيقي الممرر إليه. لذلك فإن البارامیتر الحقيقي قابل للقراءة والكتابة بدلاً من القراءة فقط في حالة التمرير بالقيمة. عندئذ أي تغيير للمتغير المحلي داخل التابع سوف يسبب التغيير نفسه للبارامیتر الحقيقي الذي مرر إليه.

مثال (11) : البرنامج التالي يقوم بعمل البرنامج في المثال (4) ولقد مررت البارامترات إلى التابع بالعنوان باستخدام المؤشرات وذلك بإضافة المؤثر * إلى نوع بيانات البارامترات في التعريف والتصريح عن التابع، وإضافة المؤثر & إلى بارامترات التابع عند استدعاء التابع :

```
# include < iostream.h >
void swap (int*, int*);
main ()
{
        int n,m;
        cout<<"enter two integer values:";
        cin >> n >> m;
        cout<<"n="<<n<<"\t"<<"m="<<m<<endl;
        swap (&n,&m);
        cout<<"n="<<n<<"\t"<<"m="<<m<<endl;
        return 0;
}
void swap (int*x,int*y)
  {
        cout <<"*x="<<*x<<"\t"<<"*y="<<*y<<endl;
        int temp=*x;
        *x=*y;
        *y=temp;
        cout<<"*x="<<*x<<"\t"<<"*y="<<*y<<endl;
  }
```

والخرج هو :

```
enter two integer values: 10    20⏎
n =10            m = 20
*x = 10          *y = 20
*x = 20          *y = 10
n = 20             m = 10
```

مثال (12) : البرنامج التالي يقوم بإدخال قيمة واستدعاء تابع لإيجاد مكعب تلك القيمة. يتم تمرير القيمة بالمؤشر.

```
# include < iostream.h >
void cube (int*);
main ()
{
        int number;
        cout<<"enter a number between 0 and 10:";
start : cin >> number;
        if ((number<0) || (number>10))
          goto start;
          cube (& number);
          cout<<"number to the power 3="<<number<<endl;
        return 0;
}
void cube (int*y)
{
        *y=*y**y**y;
}
```

4-2-9- استخدام العامل delet , New

إن الاستخدام الأهم للمؤشرات في لغة ++c هو:

1- تمرير البارامترات بين التوابع من خلال المرجع. فمن خلال المؤشرات يتم تمرير عناوين متغيرات البارامترات بدلاً من تمرير نسخ من قيم هذه المتغيرات.

2- معالجة المعلومات في المخزن المطلق أو ما يسمى بالكومة . (heap)

3- الوصول إلى البيانات والتوابع الأعضاء في صنف (class).

تجري عملية توزيع الذاكرة عادة إلى أربع مناطق عند تنفيذ برامج ++c . وهذه المناطق توزع بين :

1- شيفرة البرنامج (program code).

2- مجموعة المعطيات.

3- المكدس (stack).

4- الكومة (heap).

ينظم المكدس المعطيات أو المؤشرات بشكل صف. بحيث إن إضافة كل معلومة تدفع (pushed) من الأعلى إلى الأسفل. وعند حاجتنا لأي مجموعة منها، فإن آخر واحد من الأعلى هو الذي يخرج أولاً وهكذا. تماماً مثل مجموعة من صواني الطعام في مطعم للخدمة الذاتية ويمثل ذلك بالعبارة : الأول في الدخل الآخر في الخرج (first in last out).

أما الكومة فإنها تعمل عمل المكدس ولكنها غير منظمة مثله، حيث إنها ترتب المعطيات ترتيباً عشوائياً إدخالاً وإخراجاً ولذلك سميت بالكومة. وتعمل الأداتان new و delete على توفير المساحة الخالية من الذاكرة التي تشكل الكومة. وتكون نتيجة الأداة new الإيجابية مؤشراً رمزياً أما الأداة delete فتعيد مساحة الذاكرة الموزعة سابقاً إلى الكومة، من أجل إعادة الاستفادة منها.

الشكل العام لهاتين الأداتين:

```
data_type * pointer_name=new  data_type;
delete pointer_name;
```

يتبع الكلمة الأساسية new نوع الكائن الذي تريد تحديده حتى يعرف المترجم كمية الذاكرة المطلوبة. مثلاً new short int تحدد 2 بايت في المخزن الكومة بينما new long تحدد 4 بايت.

القيمة المرجعة من new هي عنوان ذاكرة يجب تعيينه إلى مؤشر.

يمكننا أن نكتب مثلاً :

```
unsigned short int*ppointer=new unsigned short int;
```

هذه التعليمة تجعل المؤشر ppointer يشير إلى unsigned short في الكومة. يمكننا عن طريق هذا المؤشر تعيين قيمة ولتكن 40 لذلك الحيز من الذاكرة بكتابة:

```
* ppointer = 40;
```

هذا يعني ضع 40 في المنطقة التي يشير إليها ppointer في الكومة. إذا لم تستطع new تحديد أية ذاكرة في الكومة فإنها سوف ترجع المؤشر الخالي null pointer . لذلك يجب التحقق من المؤشر بعد كل طلب لذاكرة جديدة.

يمكن تعيين عنوان ما في الذاكرة إلى مؤشر وتعيين قيمة لذلك العنوان من الذاكرة في تعليمة واحدة كأن نكتب :

```
unsigned short int * ppointer = new unsigned short int (40);
cout << *ppointer;      // ok: both ppointer and * ppointer
                          have been initialized
```

وفي هذه الحالة إذا لم تستطع new إيجاد ذاكرة كافية فإنها ترجع القيمة 0 (المؤشر الصفري) . ولتجنب ذلك نكتب التعليمات المحكمة التالية:

```
double * ptr = new double;
if (ptr = = 0)
abort( );
else
*ptr = 3.141592658;
```

هذه التعليمات تؤدي إلى الخروج من البرنامج بتنفيذ التابع ( )abort إذا كانت قيمة المؤشر المعينة بالتابع new صفرية، وإلاّ فإنه يعين القيمة 3.141592658 للموقع من الذاكرة في الكومة الذي يشير له المؤشر ptr.

لنلاحظ التعليمتين التاليتين :

```
float x = 3.14159;     // allocates named memory
float * ptr = new float (3.14159);     // allocates unnamed memory
```

التعليمة الأولى تخصص الذاكرة وقت الترجمة للمتغير x. بينما في التعليمة الثانية تخصص الذاكرة وقت التنفيذ إلى هدف غير مسمى نصل إليه من خلال ptr*.

عند الانتهاء من المساحة التي حجزت في الذاكرة بوساطة العامل new يجب استدعاء delete لحذف المؤشر. وعندها يعيد delete المساحة المحددة إلى الكومة ولاتستخدم ثانية إلاّ بعد إعادة تخصيصها.

تذكر أن المتغير المحلي مثل المؤشر يندثر ويفقد نطاقه عندما تنتهي من تنفيذ التابع ، أما الذاكرة المحددة باستعمال new فلا تخلى وتندثر تلقائياً وتظل غير متوافرة حتى إعادتها إلى الكومة باستعمال delete حيث يتم حذف مؤشرٍ. أي تحرير الذاكرة التي حفظ عنوانها في المؤشر ، كأن تقول أعد إلى الكومة كل الذاكرة التي يشير إليها هذا المؤشر.

لقد ذكرنا أن اسم المصفوفة هو مؤشراً ثابتاً يخصص له ذاكرة في وقت الترجمة:

```
float a[20];      // a is a const pointer to a block of 20 floats
float * const ptr = new float [20];
```

هنا كلٍ من a و ptr هما مؤشران ثابتان لحيز مكون من 20 عدد حقيقي. إعلان a يسمى ربط استاتيكي (static binding) لأنها مخصصة في وقت الترجمة. ويقتصر الرمز على الذاكرة المخصصة حتى ولو لم تستخدم المصفوفة في أثناء تنفيذ البرنامج.

على النقيض يمكننا استخدام مؤشر غير ثابت لتأجيل تخصيص الذاكرة حتى وقت تنفيذ البرنامج. عادة ما تسمى ربط وقت التنفيذ أو الربط الديناميكي.

```
float * ptr = new float [20];
```

والمصفوفة المعلن عنها بهذه الطريقة تسمى مصفوفة ديناميكية.

لنقارن بين الطريقتين لتعريف المصفوفة :

```
float a[20];     // static array
float * ptr = new float [20];      // dynamic array
```

المصفوفة الاستاتيكية a تكونت في وقت الترجمة. تبقى الذاكرة الخاصة بها في خلال تنفيذ البرنامج، المصفوفة الديناميكية ptr تكونت في وقت تنفيذ البرنامج . تخصص لها الذاكرة فقط عند تنفيذ إعلانها، أكثر من ذلك فإن الذاكرة المخصصة للمصفوفة ptr تحرر بمجرد استدعاء عامل الحذف delete عليها.

```
delete [ ] ptr;      //  deallocates the array ptr
```

لاحظ ضرورة وجود الأقواس [ ] بهذه الطريقة حيث إن ptr هي مصفوفة.

5-2-9- التعابير والعمليات الحسابية على المؤشرات
(Pointer expression and pointer arithmetic):

يمكن استخدام المؤشرات في التعابير الحسابية ، وتعابير التعيين ، وتعابير المقارنة ولكن يجب الانتباه إلى أنه لايمكن استخدام كافة العمليات في التعابير التي تحوي مؤشرات كمتحولات. كما أنه يجب أن تكون المؤشرات كلها في تعبير ما تشير إلى معطيات من النوع نفسه. ولمعرفة حجم أي نوع من المعطيات من المفيد استخدام التابع sizeof( ).

مثال (13) البرنامج التالي يعطي ا لحجم بالبايت الذي يشغله كل نوع من الأنواع المستخدمة في لغة c++:

```
# include < iostream.h >
main ( )
{
        int array [5];
        cout << " number of bytes used : \ n ";
        cout << " \ t char : " << sizeof (char) << endl;
        cout << " \ t short : " << sizeof (short) << endl;
        cout << " \ t int : " << sizeof (int) << endl;
        cout << " \ t long : " << sizeof (long) << endl;
        cout << " \ t float : " << sizeof (float) << endl;
        cout << " \ t double : " << sizeof (double) << endl;
        cout << " \ t long double : " << sizeof (long double) << endl;
        cout << " \ t number of bytes in array is : "
              << sizeof (array) << endl;
        return 0;
}
```

خرج البرنامج :

```
number of bytes used:
char: 1
short: 2
int: 2
long: 4
float: 4
double: 8
long double: 10
number of bytes in array is: 10
```

9-2-5-1- علاقة التعيين للمؤشرات :

يمكن استخدام المؤشر كطرف أيمن في علاقات التعيين، لتعيين قيمته إلى مؤشر آخر وذلك مع العلم أن المؤشرين يجب أن يشيرا إلى معطيات من النوع (type) نفسه

مثال (14):

```
# include < iostream.h >
main ()
{
   int n = 20;
   int * nptr1 , * nptr2;
   nptr1 = & n;
   nptr2 = nptr1;
   cout << " & n = " << & n << endl;
   cout << " nptr1 = " << nptr1 << endl;
   cout << " nptr2 = " << nptr2 << endl;
   return 0;
}
```

والخرج هو :

```
& n = 0 x 21365FD4
nptr1= 0 x 21365FD4
nptr2 = 0x 21365FD4
```

نلاحظ أن كلا المؤشرين nptr1 و nptr2 يحملان عنوان المتغير n وهو (0x21365FD4) .

9-2-5-2- العمليات الحسابية على المؤشرات:

هناك مجموعة محدودة من العمليات التي يمكن تطبيقها على المؤشرات. فمثلاً يمكن زيادة المؤشر بـ 1 باستخدام العملية (++) ، وإنقاصه بـ 1 باستخدام العملية (--)، ويمكن إضافة قيمة صحيحة إليه بوساطة العملية (+ أو = +) ، وطرح قيمة صحيحة منه بوساطة العملية (- أو = -) ، ويمكن أيضاً طرح مؤشر من مؤشر آخر.

مثال (15): البرنامج التالي يوضح إجراء العمليات الحسابية السابقة على المؤشرات:

```
# include < iostream.h >
main ( )
{
        int a = 10;
        int * aptr = & a;
        cout << " & a = " << & a << " \t and aptr = " << aptr << endl;
        ++ aptr;
        cout << " after aptr++:" << "\t aptr = " << aptr << endl;
        aptr + = 2;
        cout << " after aptr+ = 2:\t aptr = " << aptr << endl;
        aptr --;
        cout << " after aptr-- :\t aptr = " << aptr << endl;
        aptr - = 5;
        cout << " after aptr - = 5:\t aptr = " << aptr << endl;
        int * aptr2 = & a;
        cout << " aptr2 – aptr = " << aptr2 – aptr << endl;
        cout << " aptr-aptr2="<< aptr-aptr2 << endl;
        return 0;
}
```

والخرج هو :

```
& a = 0x23c72452            and aptr = 0x23c72452
```

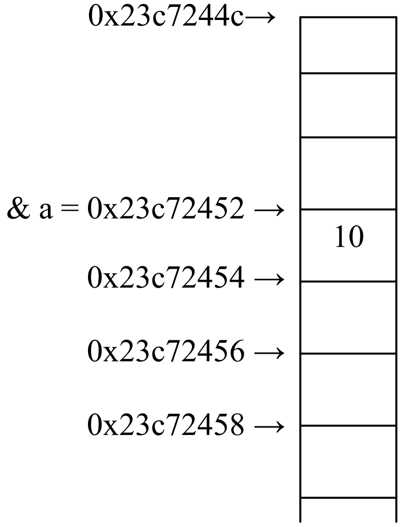
```
after aptr++:          aptr = 0x23c72454
after aptr+ = 2:       aptr = 0x23c72458
after aptr--:          aptr = 0x23c72456
after aptr- = 5:       aptr = 0x23c7244c
aptr2 – aptr = 3
aptr – aptr2 = -3
```

نلاحظ من خرج البرنامج أن زيادة المؤشر بعدد صحيح تؤدي إلى زيادة قيمته بمقدار حجم النوع الذي يشير إليه المؤشر مضروباً بالعدد الصحيح حيث عندما كان aptr=0x23c72454 وتم زيادته بـ 2 أصبحت قيمته aptr = 0x23c72458 أي زادت بمقدار 4 بايت وهي حاصل جداء 2 × حجم القيمة int وهي 2 .

كذلك الأمر بالنسبة لإنقاص المؤشر بعدد صحيح ، سينقص من قيمة المؤشر حاصل جداء القيمة المطلوب إنقاصها مضروبة بحجم النوع الذي يشير إليه المؤشر.

أما طرح مؤشر من مؤشر فإنه سيعيد عدد العناصر من النوع المشار إليه والتي يمكن تخزينها بين العنوانين اللذين يشير إليهما المؤشران.

يمكن تمثيل مخطط الذاكرة للمثال السابق كما في الشكل (9-2) التالي:



الشكل (9-2) **مخطط الذاكرة للمثال** (15)

نلاحظ مما سبق أن المؤشرات الحسابية يمكن أن تستخدم بشكل فعال مع المصفوفات لأنه لايمكن أن نفترض وجود متحولين من النوع نفسه ومتجاورين في الذاكرة إلاّ إذا كانا عنصرين متجاورين من عناصر مصفوفة.

9-2-5-3- مقارنة المؤشرات (The pointer comparison):

يمكن مقارنة مؤشرين في تعبير علائقي، وبموجب عملية المقارنة تتم مقارنة العناوين المخزنة ضمن المؤشرات، ليحدد بعدها أيهما يشير إلى موقع في الذاكرة أعلى من الآخر أو أقل. وهنا أيضاً نلاحظ إمكانية استخدام المؤشرات مع المصفوفات. حيث إن عملية المقارنة بين مؤشرين على المصفوفة نفسها يمكن أن تبين أن أحدهما يشير إلى عنصر ذي دليل أعلى أو أدنى من العنصر الذي يشير إليه المؤشر الآخر.

مثال (16):

```
# include < iostream.h >
main ( )
{
        int a = 10 , b = 20;
        int* aptr = & a;
        int* bptr = & b;
        cout << "& a=" << &a << "\t and aptr = " << aptr << endl;
        cout << "& b=" << &b << "\t and bptr = " << bptr << endl;
        if ( aptr < bptr)
        cout << " aptr points to a lower memory location"
                  << " than bptr" << endl;
        else
        cout << " bptr points to a lower memory location"
                   << "than aptr" << endl;
        return 0;
}
```

والخرج هو :

```
&a=0x0065FDF4            and aptr = 0x0065FDF4
```

&b=0x0065FDEC and bptr = 0x0065FDEC
bptr points to a lower memory location than aptr.

9-2-6- العلاقة بين المصفوفات والمؤشرات

:(The relation between pointers & arrays)

هناك علاقة وثيقة بين المؤشرات والمصفوفات في لغة ++c . بما أن العنصر الأول للمصفوفة يُعد مؤشراً أو دليلاً للمصفوفة في الذاكرة، يتعامل من خلاله معها، فإن هناك تشابهاً كبيراً بين المصفوفات والمؤشرات في طريق وصول كل منهما إلى الذاكرة، وهناك أيضاً فروق بينهما، حيث المؤشر متغير يُعد العناوين كقيم، بينما اسم المصفوفة يعد عنواناً أو مؤشراً لكنه ثابت.

9-2-6-1- المؤشرات والمصفوفات الأحادية البعد:

لو كان لدينا المصفوفة الصحيحة [10]array ، والمؤشر aptr للنوع الصحيح int . فإنه يمكننا كتابة ما يلي:

```
int array[10];
int*aptr;
```

وطالما أن اسم المصفوفة هو مؤشر على أول عنصر فيها، عندها يمكن كتابة :

```
aptr = array;
or          aptr = & array[0];
```

وهما تعليمتان متكافئتان حيث تم نسب عنوان العنصر الأول من المصفوفة array للمؤشر aptr. مما سبق نستطيع كتابة :

```
array = = & array[0];
```

ولو أردنا الوصول إلى العنصر الرابع مثلاً من المصفوفة array باستخدام تعابير المؤشرات لكان علينا كتابة ما يلي :

```
* (aptr + 3 )
```

حيث يدل الرقم 3 على الانزياح (offset) مقارنة بقيمة المؤشر aptr. مما سبق نستطيع القول إنه إذا أشر مؤشر ما على بداية مصفوفة فإن الانزياح المضاف إليه يدل

على أحد عناصر المصفوفة الذي يطابق دليله ذلك الانزياح. نسمي طريقة الكتابة هذه بترميز مؤشر/إزاحة (pointer/offset notation) . يجب الانتباه إلى ضرورة وجود الأقواس في الكتابة السابقة، ذلك أن أولوية العملية * هي أعلى من أولوية عملية الجمع + . التعبير السابق من دون أقواس يصبح:

*aptr + 3

وهذا مكافئ لإضافة القيمة 3 للتعبير aptr* (أي إضافة 3 لقيمة العنصر الأول array[0] وذلك بفرض أن المؤشر يؤشر على بداية عناصر المصفوفة).

أما لو أردنا الوصول إلى قيمة العنصر الرابع من المصفوفة باستخدام اسم المصفوفة كمؤشر لأمكننا كتابة ما يلي :

* (array + 3)

يمكن إضافة أدلة إلى المؤشرات كما نفعل تماماً مع اسم المصفوفة ونكتب التعبير:

aptr[3]

الذي يدل على العنصر array(3) نسمي طريقة الكتابة السابقة بترميز مؤشر/دليل (pointer/subscript notation).

```
int * pointer,    matrix[10];
pointer = matrix;
pointer[3] =50;
*(pointer+3)=50;
```

التعليمات السابقة تصرح عن مؤشر لقيمة صحيحة ومصفوفة صحيحة أحادية ذات 10 عناصر، عيّن المؤشر على العنصر الأول منها بالتعليمة الثانية، أما التعليمة الثالثة والرابعة فقد عينت القيمة 50 للعنصر الرابع في المصفوفة matrix أو matrix[3] . وذلك باستخدام ترميز مؤشر /دليل ثم مؤشر/ إزاحة.

تذكر دائماً وكما قلنا سابقاً إن اسم المصفوفة هو عبارة عن مؤشر ثابت وهو يؤشر دوماً على بداية المصفوفة. لذلك فالتعبير :

array+=3;

هو تعبير خاطئ لأنه يحاول تغيير قيمة اسم المصفوفة باستخدام إحدى العمليات الحسابية ، كذلك التعليمتان

```
array = aptr;
++ array;
```

تُعدان غير مقبولتين لأنه لايمكن تغيير عنوان المصفوفة (مؤشرها في الذاكرة). لأن array (الممثلة بمؤشر لها) تمثل قيمة ثابتة لاتتغير.

إذن تذكر أنه بالرغم من أن أسماء المصفوفات هي عبارة عن مؤشرات على بداية عناصرها . وعلى الرغم من إمكانية تغيير المؤشرات باستخدام التعابير الحسابية. فإنه لايمكن تغيير قيمة اسم مصفوفة أبداً باستخدام التعابير الحسابية . لو كان لدينا:

```
const int size = 10;
 int array [size];
int * pointer = array;
```

وأردنا حساب مجموع عناصر المصفوفة array عن طريق المؤشر pointers. لاستطعنا ذلك عن طريق التعليمات التالية :

```
 int Sum=0;
 for (pointer=array; pointer< & array [size]; pointer++)
Sum +=* pointer;
```

في حلقة التكرار السابقة يتم تعيين قيمة للمؤشر الأول وهي عنوان عنصر المصفوفة الأول array[0]، ومن ثم فإن قيم المؤشر pointer التالية تكافئ & array[2], & array[1] .... الخ

وبشكل عام ، لو كان العداد i من النوع الصحيح، فإن pointer+i يمثل إزاحة في قيمة العنوان للمؤشر pointer بمقدار i ، وبطريقة مشابهة، فإن array+i يمثل إزاحة من العنوان الأول للمصفوفة. وهذه طريقة أخرى لجمع عناصر المصفوفة.

```
Sum=0;
for(int i=0; i<size; i++)
    Sum+=*(array+i);
```

مثال (17): البرنامج التالي يستخدم مؤشراً للتنقل بين عناصر المصفوفة array:

```
# include < iostream.h >
main ()
{
        const int size = 3;
        short sum = 0;
        short array [size] = { 10 , 20 , 30 };
        cout << "array=" << array << endl;
        cout << "sizeof(short)="<<sizeof (short)<< endl;
        short*end = array + size;      // convert size to offset 6
        for (short*ptr = array; ptr < end; ptr++)
        {
              sum + = * ptr;
              cout << "\t ptr = " << ptr;
              cout << "\t* ptr = " << *ptr;
              cout << "\t sum = " << sum << endl;
        }
        cout << "end=" << end << endl;
        return 0;
}
```

والخرج هو :

```
array = 0x238f2424
sizeof(short)=2
        ptr=0x238f2424          *ptr=10              sum=10
        ptr=0x238f2426          *ptr=20              sum=30
        ptr=0x238f2428          *ptr=30              sum=60
end = 0x238f242a
```

مثال (18): البرنامج التالي يستخدم اسم المصفوفة مع إزاحة للدلالة على عناصر المصفوفة حيث إن عامل الفهرس للمصفوفة [ ] يكافئ عامل إعادة المرجعية * أي أن:

```
        array [0] = =* array;
        array[1] = =* (array+1);
        array[2] = =* (array+2);
# include < iostream.h >
main( )
{
        short array [ ] = { 10 , 20 , 30 , 40 , 50 };
        cout <<"array= "<<array<<"\t * array="<<*array<<endl;
        for(int i=0;  i<5;  i++)
            cout<<"array+"<<i<<"="<<array+i<<"\t"
                <<"*(array+"<<i<<")="<<*(array+i)
                << endl;
        return 0;
}
```

والخرج هو :

```
array = 0x24cf2418            * array = 10
array + 0 = 0x24cf2418                 * (array+0)=10
array + 1 = 0x24cf241a                 * (array+1)=20
array + 2 = 0x24cf241c                 * (array+2)=30
array + 3 = 0x24cf241e                 * (array+3)=40
array + 4 = 0x24cf2420                 * (array+4)=50
```

مثال (19): البرنامج التالي يطبع عناصر المصفوفة الأحادية البعد باستخدام طرق الترميز الأربعة التي تعرفنا عليها سابقاً:

```
// using subscripting and pointers with arrays
# include < iostream.h >
main ( )
{
        int array [ ] = { 15 , 20 , 30 , 10 };
        int * aptr = array;     // set aptr to point to array
        cout << " array printed with:" << endl
              << " array subscripted notation:" << endl;
        for ( int i=0; i<=3; i++)
            cout << "array [ "<<i<<"]="<< array [i] <<endl;
            cout<<endl<< "pointer/offset notation where:"<<endl
                 <<"the pointer is the array name:" << endl;
            for ( i = 0 ; i < = 3; i++)
                 cout << " * (array+" << i << ") = "
                        << * (array+i) << endl;
        cout << endl << " pointer subscript notation:" << endl;
        for ( i=0;  i<=3; i++)
            cout << " aptr [ " << i <<" ] = " << aptr [i] << endl;
        cout << endl << " pointer/offset notation:" << endl;
            for ( i=0; i<=3; i++)
                 cout << " * (aptr + " << i << " ) ="
                       << * (aptr+i) << endl;
        return 0;
}
```

والخرج هو :

```
array printed with:
array subscripted notation:
array [0] = 15
array [1] = 20
array [2] = 30
array [3] = 10
pointer/offset notation where
the pointer is the array name:
* (array+0) = 15
* (array+1) = 20
* (array+2) = 30
* (array+3) = 10
pointer subscript notation:
aptr[0] = 15
aptr[1] = 20
aptr[2] = 30
```

aptr[3] = 10
pointer/offset notation:
* (aptr+0) = 15
* (aptr+1) = 20
* (aptr+2) = 30
* (aptr+3) = 10

2-6-2-9- المؤشرات والمصفوفات ثنائية البعد:

بطريقة مماثلة لو كان لدينا المصفوفة array ذات بعدين. فإن التعليمة التالية صحيحة:

array = =& array[0][0];

حيث array مؤشر اسم المصفوفة، وهو مؤشر ثابت يؤشر على العنصر الأول دوماً .

& array [0][0] مؤشر العنصر الأول.

كذلك يمكننا الحصول على عنوان أول عنصر من المصفوفة ثنائية البعد array بكتابة ما يلي :

*(array)

لو كان لدينا المصفوفة ثنائية البعد array[n][m] حيث n عدد الأسطر ، وm عدد الأعمدة. فإنه يمكننا الحصول على عنوان العنصر array[i][j] حيث i هو عداد الأسطر. و j هو عداد الأعمدة من خلال التعبير التالي:

*(array)+(i*row_length)+j

وبشكل عام يمكننا أن نكتب المعادلة التالية في حالة مصفوفة ثنائية البعد:

*array[i][j] = * (*(array)+(i*row_length)+j);

حيث row_length هو طول سطر المصفوفة الثنائية وهو نفسه عدد الأعمدة m.

العلامة * بعد علامة التعيين (=) هي علامة المؤشر، بينما علامة * بعد علامة الجمع (+) هي علامة الضرب. ومن المهم التذكير أن عناصر المصفوفة ثنائية البعد تخزن في الذاكرة في أماكن متجاورة كما لو كانت عناصر تلك المصفوفة تنتمي لمصفوفة أحادية عدد عناصرها n*m حيث n عدد الأسطر و m عدد الأعمدة في المصفوفة الثنائية البعد.

يمكن أيضاً تمثيل العنصر الرابع (0,3) بطريقتين:

1- بوصف قيمتي العدادين للأسطر والأعمدة array [0] [3]

2- بوساطة المؤشر (array+3)*

لاحظ هنا أن هناك إزاحة بمقدار 3 عن المؤشر الأول للعنصر الأول.

مثال (20): البرنامج التالي يدخل قيماً لمصفوفة ثنائية num[2][3] ثم يطبع عناوين عناصرها وقيمها بالاعتماد على اسم المصفوفة /دليل واسم المصفوفة كمؤشر/إزاحة.

```
# include < iostream.h >
main ()
  {
        const int n = 2;
        const int m = 3;
        int num[n] [m];
        cout<<"enter num["<<n<<"]["<<m<<"]"<<endl;
        for (int i=0; i<n; i++)
              for (int j=0; j<m; j++)
                   cin >> num[i][j];
        cout<<" num= "<<num<<"\t* num ="<<*num
              <<"\t** num ="<<** num <<endl;
        for (i=0; i<n; i++)
          for (j=0; j<m; j++)
             cout <<" num ["<<i<<"]["<<j<<"]="<< num [i][j]
                  <<",  * num +i*m+j="<<* num +i*m+j
                  <<" , *(* num +i*m+j)="<<*(* num +i*m+j) <<endl;
      return 0;
 }
```

والخرج هو :

```
enter num[2][3]
22    33    44   55    66    77↵
```

```
num=0x2f672504        *num=0x2f672504        **num=22
num[0][0]=22  ,  *num+i*m+j=0x2f672504   ,*(*num+i*m+j)=22
num [0][1]=33  ,  * num +i*m+j=0x2f672506  ,  *(* num +i*m+j)=33
num [0][2]=44  ,  * num +i*m+j=0x2f672508  ,  *(* num +i*m+j)=44
num [1][0]=55  ,  * num +i*m+j=0x2f67250A  ,  *(* num +i*m+j)=55
num [1][1]=66  ,  * num +i*m+j=0x2f67250c  ,  *(* num +i*m+j)=66
num [1][2]=77  ,  * num +i*m+j=0x2f67250e  ,  *(* num +i*m+j)=77
```

لقد تم إدخال القيم التالية:77, 66 , 55 , 44 , 33 , 22 لعناصر المصفوفة num نتيجة لتنفيذ حلقتي الإدخال for i و for j . وتم طباعة عنوان العنصر الأول باستخدام اسم المصفوفة num أولاً ثم باستخدام num * ثانياً، أما قيمته فطبعت باستخدام num ** . في كل سطر من الأسطر الستة التالية تم طباعة اسم العنصر وقيمته بترميز اسم المصفوفة/ دليل ، ثم طبع عنوان العنصر بصيغة تستخدم اسم المصفوفة/إزاحة لحسابه ، وكذلك تم طباعة قيمة العنصر باستخدام ترميز اسم المصفوفة كمؤشر/إزاحة.

9-2-7- السلاسل والمؤشرات:

لنعيد كتابة التصريح عن السلاسل باستخدام ترميز المصفوفات ثم باستخدام ترميز المؤشرات

```
char str1[ ] = "c++ programming language";   //defined as an array
char *strptr[ ] = "Computer Departement";   //defined as a pointer
```

التعليمة الأولى تصرح عن سلسلة محرفية كمصفوفة بينما التعليمة الثانية تصرح عن سلسلة محرفية باستخدام المؤشرات:

التعليمتين التاليتين :

```
cout <<str1 << endl;
cout <<strptr << endl;
```

كلاهما يظهر سلسلة المحارف المقابلة.

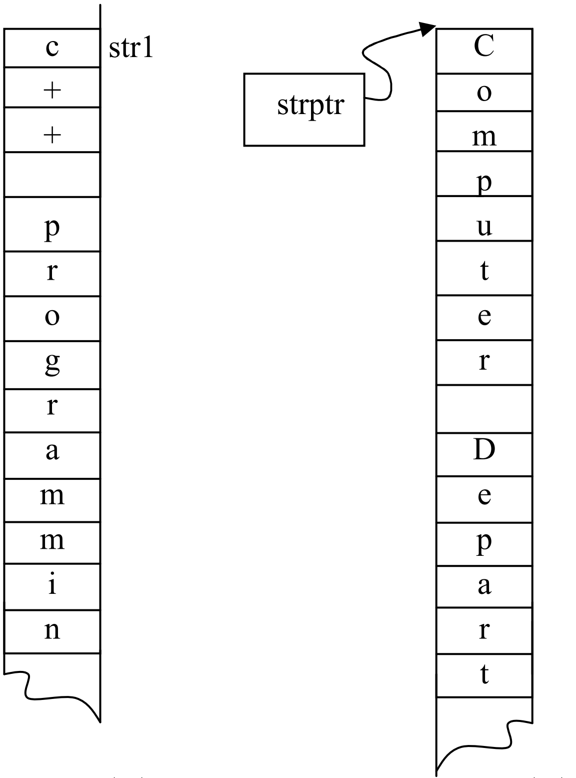
لو كتبنا :

```
str1++;       // can not do this; str1 is a constant
strptr++;    // This is ok, strptr is a pointer
```

هذان النوعان من الثوابت السلسلية مترادفان في عدة طرق. يمكنك عرضهما، واستعمالهما كباراميترات لتوابع ... الخ لكن هناك فرق : str1 هو عنوان أي ثابت مؤشر، بينما strptr هو متغير مؤشر، كما ذكرنا سابقاً عن اسم المصفوفة والاختلاف بينه وبين المؤشر إلى مصفوفة لذا يمكن تغيير strptr على عكس str1 الذي لايمكن تغييره.

يبين الشكل (3-9) كيف يبدو هذان النوعان من السلاسل في الذاكرة.

لايمكن زيادة str1 لأنه مؤشر ثابت بينما زيادة strptr ممكنة لأنه مؤشر متغير وبعد أن تتم الزيادة لن يعود ليؤشر على بداية السلسلة الحرفية.



سلسلة معرفة كمؤشر سلسلة معرفة كمصفوفة

char*strptr="Com... char str1[ ]="c++ pro...

الشكل (3-9) السلاسل كمصفوفات ومؤشرات

9-2-7-1- السلاسل كباراميترات للتوابع

نبين في المثال التالي تمرير سلسلة إلى تابع يقوم بإظهار هذه السلسلة .

مثال (21) : البرنامج التالي يصرح عن سلسلة محرفية ثم يمررها إلى تابع يقوم بطباعتها حرفاً حرفاً حتى الوصول إلى نهايتها.

```
# include < iostream.h >
void dispstr (char*);
main ( )
{
        char str[ ] = "c++ Language";
        dispstr(str);       //    display the string.
        return 0;
}
void dispstr(char*ptr)
{
        cout<<endl;                 // start on new line
        while (*ptr)                // until null character
          cout<< *ptr++;           // print character
}
```

يتم استعمال عنوان المصفوفة str كباراميتر في استدعاء التابع ( )dispstr. هذا العنوان هو ثابت لكن بما أنه ممرر بالقيمة، يتم إنشاء نسخة عنه في التابع ( )dispstr، هذه النسخة هي مؤشر يدعى ptr. يمكن تغيير المؤشر لذا يقوم التابع بزيادة ptr من أجل عرض كل حرف في السلسلة. يعيد التعبير ++ptr* الأحرف المتتالية. تتكرر الحلقة إلى أن نجد الحرف الخامد ('0\') في نهاية السلسلة. بما أن هذا ا لحرف قيمته 0. أي خطأ ، تتوقف الحلقة while عنده.

مثال (22): البرنامج التالي يستخدم تابعاً يقوم بنسخ سلسلة إلى سلسلة أخرى:

```
# include < iostream.h >
void copystr (char*,char*);          // Function prototype
main ()
{
    char * str1 = "c++ programe";
    char  str2[20];                  // empty string.
    copystr (str2,str1);             // copy str1 to str2
    cout << endl << str2;            // display str2
    return 0;
}
```

```
void copystr (char*dest,char*sour)
{
    while (*sour)                       // until null character
   *dest++=*sour++;              // copy chars from sour to dest
   *dest = "\0";                     // terminate dest.
}
```

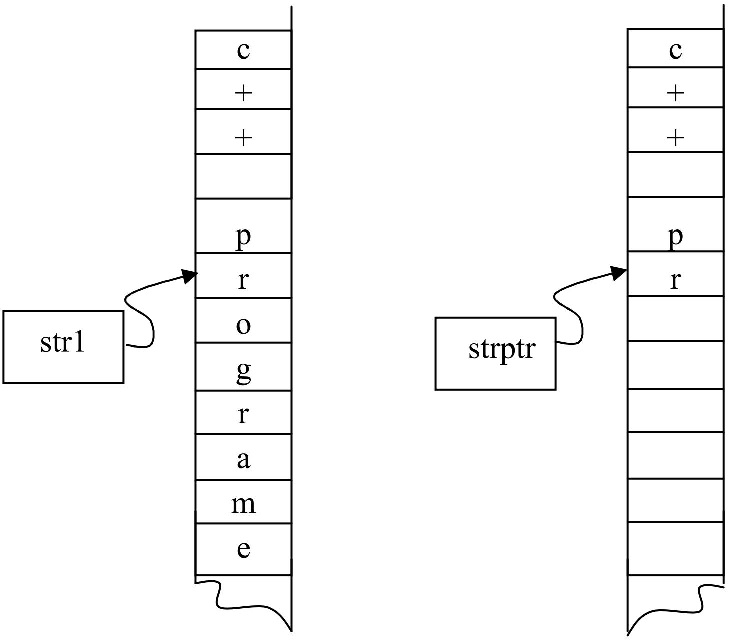
الخرج هو :

c++ programe

يستدعي التابع main التابع copystr( ) لنسخ السلسلة str1 إلى السلسلة str2التعبير:

*dest++ = * sour++;

يأخذ قيمة الحرف من العنوان الذي يشير إليه sour ويضعه في العنوان الذي يشير له dest. ثم تتم زيادة المؤشرين. لذا سيتم نسخ الحرف التالي في التكرار التالي للحلقة. تنتهي الحلقة عند العثور على Null في sour. عندها يتم وضع الحرف '\0' في dest ويعود التنفيذ إلى التابع main( ) لتتم طباعة السلسلة str2 . الشكل (4-9) يبين كيف تنتقل المؤشرات في السلاسل.



*dest++=*sour++;

الشكل (9-4) نسخ سلسلة باستعمال المؤشرات

9-2-8- استخدام الواصف const مع المؤشرات

(Using the const qualifier with pointers)

إن استخدام الواصف const يعني عدم السماح بتغيير قيمة متغير محدد ويجعل المترجم يلتقط ذلك ويمنع القيام بالتغيير مرسلاً رسالة تحذير أو خطأ. ويمكن استخدام المؤشر مع const في أربعة أشكال:

1- مؤشر غير ثابت لمعطيات غير ثابتة.

2- مؤشر غير ثابت لمعطيات ثابتة.

3- مؤشر ثابت لمعطيات غير ثابتة.

4- مؤشر ثابت لمعطيات ثابتة.

9-2-8-1- المؤشر غير الثابت لمعطيات غير ثابتة:

(Non_constant pointer to non_constant data):

نحصل على أعلى مستوى للوصول بوساطة مؤشر غير ثابت على معطيات غير ثابتة. أي أنه يمكننا تغيير المعطيات باستخدام المؤشرات عليها ويمكن أيضاً تغيير وجهة المؤشر ليؤشر على معطيات أخرى. لايحوي التصريح عن مؤشر غير ثابت على معطيات غير ثابتة على الواصف const.

مثال (23): البرنامج التالي يعين مؤشراً على مصفوفة ويتزايد ليمسح عناصر المصفوفة ويقوم بمضاعفة قيمها وهذا ما يوضح إمكانية تغيير المعطيات ومكان تأشير المؤشر باستخدام مؤشر غير ثابت لمعطيات غير ثابتة.

```
# include < iostream.h >
void change_element(int*);
main ()
{
        int array [ ] = { 5 , 10 , 7 , 13 , 20 , 4 };
        cout << " display array : " << endl;
        for (int i=0; i<6; i++)
        cout << array [i] << "   ";
        cout << endl;
        change_element(array);
        cout << " array after the change is: " << endl;
        for (i=0; i<6; i++)
              cout << array [i] << "   ";
              cout << endl;
        return 0;
}
void change_element (int*ptr)
{
        for (int i=0; i<6; i++)
        {
              *ptr=2*(*ptr);
               ptr++;
        }
}
```

والخرج هو :

```
display array:
5    10    7    13    20    4
array after the change is:
10    20    14    26    40    8
```

9-2-8-2- المؤشر غير الثابت لمعطيات ثابتة:

(Non_constant pointer to constant data):

تستخدم المؤشرات غير الثابتة على معطيات ثابتة وذلك بهدف تمرير المعطيات إلى تابع مثلاً وطباعتها أو البحث عن عنصر فيها من دون تغييرها. كما يمكن تغيير

المؤشر ليؤشر على معطيات أخرى لها النوع نفسه ولكن لايمكن تغيير المعطيات المؤشر عليها.

لو أعدنا كتابة البرنامج السابق وغيرنا تعليمة التصريح عن المؤشر ptr إلى الشكل التالي :

```
const int * ptr ;
```

وبتنفيذ البرنامج فإن المترجم يعطي رسالة خطأ تشير إلى عدم إمكانية تغيير المعطيات الثابتة.

إذن بتنفيذ البرنامج التالي :

مثال (24):

```
# include < iostream.h >
void change_element(const int*);
main ()
{
        int array [ ] = { 5 , 10 , 7 , 13 , 20 , 4 };
        cout << " display array : " << endl;
        for (int i=0; i<6; i++)
            cout << array [i] << "   ";
        cout << endl;
        change_element(array);
        cout << " array after the change is: " << endl;
        for (i=0; i<6; i++)
              cout << array [i] << "   ";
        cout << endl;
        return 0;
}
void change_element (const int*ptr)
{
        for (int i=0; i<6; i++)
        {
              *ptr=2*(*ptr);
            ptr++;
        }
}
```

سيعطي المترجم رسالة تعلمنا بأنه لايمكن تغيير قيمة ثابتة في التابع:

```
Compiling Ex8_24
Error Ex8_24.cpp2:Cannot modify a const object in function
                        change-element (const int*)
```

مثال (25): البرنامج التالي يستخدم مؤشراً متغيراً لمعطيات ثابتة بحيث يتغير المؤشر ليمسح عناصر المصفوفة ويطبعها ولكنها تبقى ثابتة:

```
# include < iostream.h >
void printarray(const int*);
main ()
{
        int array [ ] = { 10 , 20 , 30 , 40 , 50 , 60 };
        cout << " display array : " << endl;
        printarray (array);
        return 0;
}
void printarray (const int*ptr)
{
        for (int i=0; i<6; i++)
        {
            cout << *ptr<< "   ";
            ptr++;
        }
        cout << endl;
}
```

والخرج هو :

```
display array:
10   20   30   40   50   60
```

3-8-2-9- المؤشر الثابت على معطيات غير ثابتة:

(Constant pointer to non_constant data):

المؤشر الثابت على معطيات غير ثابتة يشير إلى الموقع نفسه من الذاكرة الذي يمكن أن تتغير المعطيات المخزنة في ذلك الموقع باستخدام المؤشر. وتماثل هذه الحالة حالة اسم المصفوفة الذي هو عبارة عن مؤشر ثابت على أول عنصر من عناصرها. ويمكن الوصول إلى أي عنصر من عناصر المصفوفة وتغييره باستخدام اسم المصفوفة ودليل هذا العنصر.

تستخدم المؤشرات الثابتة على المعطيات غير الثابتة لتلقي مصفوفة على شكل وسيط لتابع يقوم بالوصول إلى عناصرها باستخدام أدلة هذه العناصر. وعند التصريح عن مؤشر أنه const يجب إعطاؤه قيمة ابتدائية (إذا كان المؤشر وسيطاً لتابع يتم إعطاؤه قيمة ابتدائية باستخدام مؤشر يُراد تمريره إلى ذلك التابع).

مثال (26) : البرنامج التالي يعمل عمل البرنامج في المثال (23) لكن هنا يتم الإعلان عن ptr على أنه مؤشر ثابت لمعطيات غير ثابتة بالشكل :

int* const prt;

وبالتالي لايمكن زيادة قيمة المؤشر ليمسح عناصر المصفوفة وإنما يجب استخدام أدلة العناصر أو فهارسها لهذا الغرض. التصريح عن المؤشر بهذا الشكل يسمح بتغيير المعطيات.

```
# include < iostream.h >
void change_element(int*const);
main ( )
{
        int array [ ] = { 5 , 10 , 7 , 13 , 20 , 4 };
        cout << " display array : " << endl;
        for (int i=0; i<6; i++)
            cout << array [i] << "   ";
        cout << endl;
        change_element(array);
        cout << " array after the change is: " << endl;
        for (i=0; i<6; i++)
              cout << array [i] << "   ";
              cout << endl;
        return 0;
}
void change_element (int*const ptr)
{
        for (int i=0; i<6; i++)
        {
              *(ptr+i)=2**(ptr+i);
        }
}
```

وخرج البرنامج هو :

```
display array:
5   10   7   13   20   4
array after the change is:
10   20   14   26   40   8
```

4-8-2-9- المؤشر الثابت لمعطيات ثابتة (Const pointer to const data):

المؤشر الثابت على معطيات ثابتة يعني أن المؤشر يؤشر دوماً على الموقع نفسه من الذاكرة وتبقى المعطيات في ذلك الموقع ثابتة لاتتغير. لذا فهو يؤمن أدنى إمكانيات للوصول للمعطيات. إحدى تطبيقات هذا النوع هو تمرير عناصر المصفوفة إلى تابع يقوم باستعراض عناصرها بالاعتماد على الأدلة ولايستطيع تغيير قيم هذه العناصر كما هو الحال عند طباعة عناصر المصفوفة أو البحث عن قيمة ضمنها ... الخ.

مثال (27): البرنامج التالي يعين قيماً لعناصر المصفوفة array ثم يدخل قيمة للمتغير m ويطلب من تابع فرعي search أن يبحث عن قيمة مماثلة لها ضمن عناصر المصفوفة الممررة له. ثم يطبع في البرنامج الرئيسي العبارة المناسبة فيما لو وجد قيمة مطابقة لها أم لا.

```
# include < iostream.h >
```

```
int search(const int*const, int&,int);
main ()
{
        int array [ ] = { 11 , 22 , 33 , 44 , 55 , 66 };
        int m , k;
        while(1)
        {
          int n = 0;
          cout<<"enter search value:";
          cin >> m;
          n = search (array , k , m);
          if (n = =1)
              cout << " element m="<<m<<"\t"
                    <<"is found at index = " << k << endl;
          else
              cout << " element m = " << m << "\t"
                    << " is not found in array " << endl;
          char str;
          cout << " enter y to continue, n to exit:";
          cin >> str;
          if (str = = 'y')
              continue;
          else
              break;
        }
        return 0;
}
int search (const int * const ptr, int  & z, int y)
{
        int x = 0;
        for (int i=0; i<6; i++)
              if (*(ptr+i)= =y)
              {
                  x=1;
```

```
                z=i;
            }
        if (x= =1)
        return 1;
        else
        return 0;
}
```

والخرج هو :

```
enter search value : 44↵
element m=44                is found at index = 3
enter y to continue, n to exit: y↵
enter search value : 12↵
element m=12                is not found in array
enter y to continue, n to exit: n↵
```

مثال (28): البرنامج التالي يستعمل مؤشراً ثابتاً لمعطيات ثابتة في طباعة عناصر المصفوفة array.

```
# include < iostream.h >
void printarray(const int*const);
main ()
{
        int array [ ] = { -5 , 10 , 12 , 7 , 15 , 9 };
        cout << " display array : " << endl;
        printarray (array);
  return 0;
}
void printarray (const int*const ptr)
{
        for (int i=0; i<6; i++)
              cout << *(ptr+i) << "    ";
        cout << endl;
}
```

والخرج هو :

```
display array:
-5   10   12   7   15   9
```

مثال (29) : البرنامج التالي يعلن عن أربع متغيرات : مؤشر ptr ، مؤشر ثابت cptr، مؤشر لثابت ptrc ، ومؤشرثابت cptrc لثابت. ويقوم ببعض العمليات عليها ونلاحظ التعليق المناسب عند كل عملية.

```
# include < iostream.h >
main ()
{
        int n=44,*ptr=&n;    // an int variable n and a pointer to an int
        cout <<"n="<< n <<"\t * ptr="<<*ptr<<endl;
        cout<<"&n="<<&n<<"\t and ptr="<<ptr<<endl;
        cout<< "ptr is non_const pointer to non_const data:"<< endl
              << "which is n so ptr and n can be change"<< endl;
        ptr++;          // ok:increments pointer ptr
        n++;      // ok: increments variable n
        cout<<"after ptr++:ptr="<<ptr<<endl;
        cout<<"after n++: n="<<n<< endl;
        int * const cptr = & n;    // a const pointer to an int
        cout << "n= " <<n<<"\t and*cptr="<<*cptr<<endl;
        cout<<"&n="<<&n<<"\t cptr="<<cptr<<endl;
        cout<<"cptr is a const pointer to non_const data"<< endl
              <<"which is n so only n can be changed. "<<endl;
        (*cptr)++;          // ok: increments int*cptr
        n++;           // ok: increments int object
        // cptr++; : Error : L value specifies const object
        cout <<" after n++: n="<<n<< endl;
        const int m=80;       // a const int
```

```
    const int * ptrc = &m;    // non_const pointer to a const int
    cout<<"m="<<m<<"\t*ptrc"<<*ptrc<<endl;
    cout<<"ptrc is non_const pointer to const data "<<endl;
    cout<<"which is m so only ptrc can be changed."<<endl;
    ptrc++;        // ok: increments pointer ptrc
    // (*ptrc)++;    illegal : int*ptrc is const
    const int*const cptrc=&m;    // const pointer to const int
    // (*cptrc)++   illegal: int*cptrc is const
    // cptrc++;  illegal:pointer cptrc is const
    cout <<"m="<<m<<"\t*cptrc="<<*cptrc<<endl;
    cout<<"cptrc is a const pointer to const data"<<endl;
    cout<<"which is m so cptrc and m can not be changed"<<endl;
    return 0;
}
```

وخرج البرنامج هو :

```
n=44          *ptr=44
&n=0x248f24f0          and ptr=0x248f24f0
ptr is non_const pointer to non_const data:
which is n so ptr and n can be changed
after ptr++: ptr=0x248f24f2
after n++: n=45
n=45          and *cptr=45
&n=0x248f24f0               cptr=0x248f24f0
cptr is a const pointer to non_const data
which is n so only n can be changed.
after n++: n = 47
m = 80             * ptrc = 80
ptrc is non_const pointer to const data
which is m so only ptrc can be changed.
m = 80             *cptrc=80
cptrc is a const pointer to const data
which is m so cptrc and m can not be changed.
```

9-2-9- مصفوفات المؤشرات (Arrays of pointers):

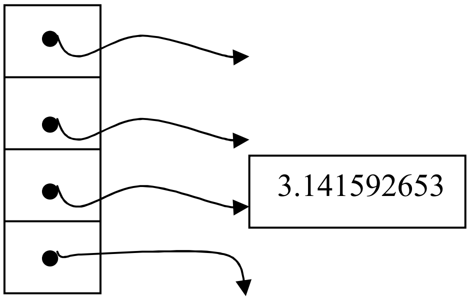
يمكن لعناصر مصفوفة ما أن تكون مؤشرات. وللتصريح عن مصفوفة من 4 عناصر كل عنصر فيها هو مؤشر لعدد من النوع double نكتب:

```
double * arrptr[4];
```

يمكن تعيين قيم لعناصرها مثل أي مؤشر آخر . لتعيين القيمة (3.141592653) للعنصر الثالث نكتب :

```
arrptr[2] = new double (3.141592653);
```

يمكن رؤية هذه المصفوفة كالتالي:



كما يمكن تعيين عنوان متغير var1 من النوع double إلى العنصر الرابع من هذه المصفوفة كما يلي:

```
double var1=5.124537;
arrptr [3] = & var1;
```

ولطباعة قيمة المتغير var1 باستخدام مؤشره نكتب:

```
cout<< *arrptr[3];
```

لو كان لدينا المصفوفة array[4] من النوع double. وأردنا أن تشير عناصر مصفوفة المؤشرات arrptr[4] إلى عناصر المصفوفة array لأمكن كتابة ما يلي :

```
arrptr[0] = & array [0];
```

أو :

```
arrptr[0] = array;
```

على اعتبار أن اسم المصفوفة هو مؤشر ثابت ويؤشر دوماً على أول عنصر فيها .

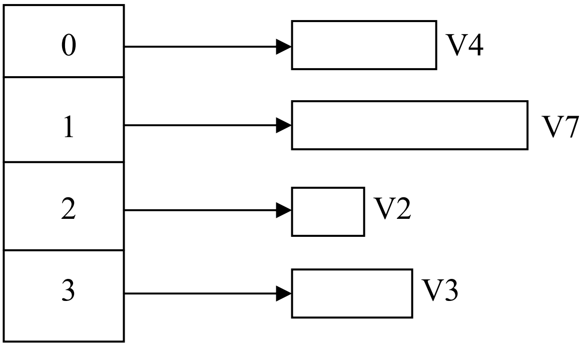
لو كان لدينا المصفوفات التالية من النوع double:

V3[3] , V7[7] , V2[2] , V4[4]

وأردنا أن تشير عناصر مصفوفة المؤشرات arrptr[4] على هذه المصفوفات وعلى أول عنصر من كل منها لأمكن كتابة ما يلي :

```
arrptr[0] = V4;
arrptr[1] = V7;
arrptr[2] = V2;
arrptr[3] = V3;
```

الشكل (9-5) يوضح العلاقة بين مصفوفة المؤشرات arrptr والمصفوفات الأخرى:



الشكل (9-5) **العلاقة بين مصفوفة المؤشرات** arrptr **والمصفوفات** V3 , V2 , V7 , V4

يمكن طباعة المصفوفة V7 بالاستدعاء التالي:

```
printrow(arrptr[1] , 7);
```

حيث التابع printrow الذي يطبع عناصر المصفوفة V7 هو التالي :

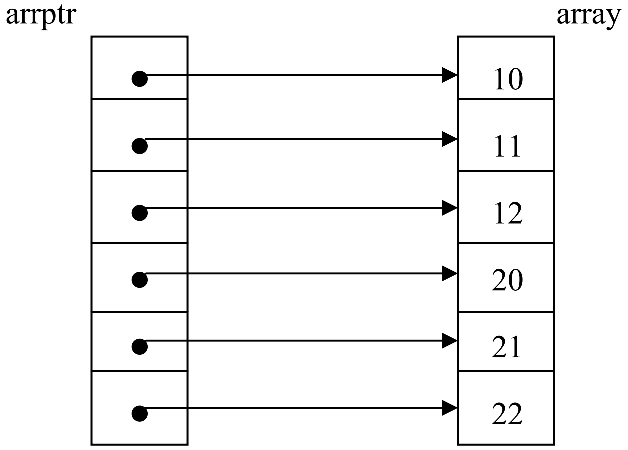
```
void printrow(double*pv, int n)
{
    for (int k=0; k<n; k++)      // for every element
        cout <<*pv++<<"   ";        // print and advance.
}
```

ولتمرير مصفوفة المؤشرات إلى تابع نستخدم الطريقة المتبعة نفسها بالنسبة إلى باقي المصفوفات، حيث يتم الاستدعاء بذكر اسم المصفوفة من دون دليل.

مثال (30) : البرنامج التالي يعين قيماً ابتدائية لمصفوفة array صحيحة، ثم يصرح عن مصفوفة من المؤشرات arrptr خصصت لعناصرها عناوين المصفوفة الصحيحة array. ومن ثم يطبع قيم عناصر المصفوفة array باستخدام عناصر مصفوفة المؤشرات arrptr.

```
# include < iostream.h >
main()
{
      int array [ ] = { 10 , 11 , 12 , 20 , 21 , 22 };
      int*arrptr[6];
      for (int i=0; i<6; i++)
          arrptr[i] = & array [i];
      cout << "display array elements using arrptr element:"
            << endl;
      for (i=0; i<6; i++)
          cout << * arrptr[i] << "   ";
      cout << endl;
      return 0;
}
```

ويمكن تصور مصفوفة المؤشرات arrptr والمصفوفة array كما في الشكل (6-9) التالي :



الشكل (6-9) يوضح العلاقة بين المصفوفة array ومصفوفة المؤشرات arrptr

مثال (31) : البرنامج التالي يصرح عن مصفوفة صحيحة ومصفوفة مؤشرات لقيم صحيحة ثم يستدعي التابع enter_array لإدخال قيم لعناصر المصفوفة والتابع print_array لطباعة قيم العناصر ثم التابع sort لترتيب عناصر المصفوفة بطريقة غير مباشرة بتغيير المؤشرات للعناصر بدلاً من تحريك العناصر نفسها.

```
# include < iostream.h >
void enter_array (int [ ], int &);
void print_array (int* [ ], int);
void sort (int* [ ], int);
const int size = 50;
main ()
{
    int array [size] , m;
    int*arrptr[size];
    cout<<"enter array element, 0 to end:"<<endl;
    enter_array(array,m);
    for (int i=0; i<m; i++)
        arrptr[i] = & array[i];
    cout<<"display entered array elements:"<<endl;
    print_array (arrptr, m);
    sort (arrptr,m);
    cout<<"display array after sort:"<<endl;
    print_array (arrptr,m);
    return 0;
}
void enter_array (int a[ ], int &n)
{
    for (n=0; n<size; n++)
    {
        cin>> a[n];
        if (a[n] = = 0)
          break;
    }
}
void print_array (int*p[ ], int n)
{
     for (int i=0; i<n; i++)
          cout << * p[i] <<"   ";
     cout << endl;
}
void sort ( int* p[ ], int n)
{
     int * temp;
     for (int i=0; i<n; i++)
          for (int j=0; j<n-1; j++)
               if (* p[j]>*p[j+1])
                    {
                         temp = p[j];
                         p[j] = p[j+1];
                         p[j+1] = temp;
                    }
}
```

خرج البرنامج هو :

```
enter array elements, 0 to end:
6   5   3   9   4   8   0
display entered array elements:
6   5   3   9   4   8
display array after sort:
3   4   5   6   8   9
```

لقد أعلن في البرنامج عن مصفوفة للأعداد الصحيحة array وعن مصفوفة من المؤشرات لأعداد صحيحة arrptr. و تم إدخال قيم عناصر المصفوفة array من خلال التابع enter_array, تنتهي عملية الإدخال عند إدخال 50 عنصراً، الصفر ليس منها، أو عند إدخال القيمة 0. ومن ثم تم تعيين عناوين عناصر المصفوفة array لمؤشرات

مصفوفة المؤشرات arrptr ثم طبعت عناصر المصفوفة array باستدعاء التابع print_array . عملية الترتيب لعناصر المصفوفة تمت باستدعاء التابع sort الذي قام بتغيير مؤشرات العناصر بدلاً من العناصر نفسها.

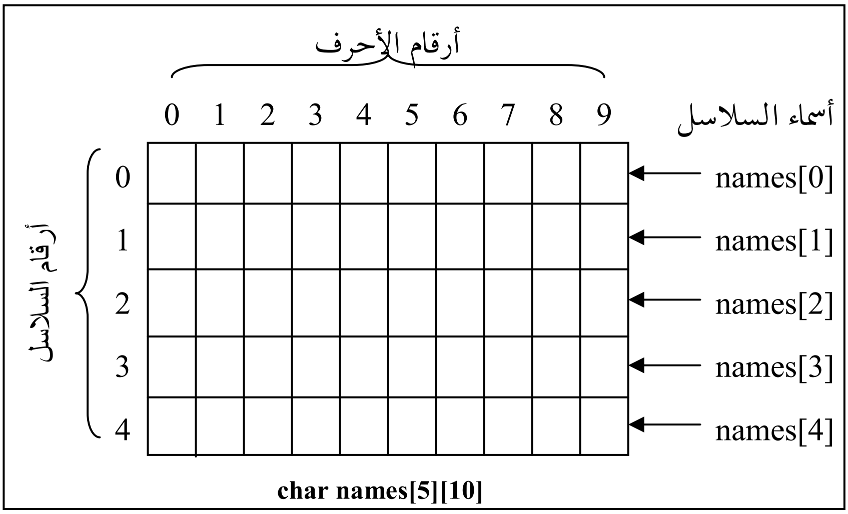
تحدثنا في الفقرة (7-6) من الفصل السابع عن المصفوفات والسلاسل. ومصفوفات السلاسل شائعة الاستخدام لأننا نستطيع بمثل هذه المصفوفة تخزين لوائح بأسماء الموظفين وكلمات المرور وأسماء الملفات وغيرها.

وكما ذكرنا تُعد مصفوفة السلاسل مصفوفة ثنائية حيث كل عنصر من عناصر مصفوفة السلاسل عبارة عن مصفوفة سلسلية أخرى.

لنفرض أننا نريد إنشاء مساحة لتخزين مجموعة من السلاسل ولتكن 5 سلاسل، كل واحدة منها يمكنها أن تخزن 10 أحرف كحد أقصى. تعليمة التصريح التالي تقوم بالعمل :

```
char names[5] [10];  //   array of 5 strings
```

لاحظ أن عدد السلاسل هو دائماً البعد الأول للمصفوفة، بينما طول كل سلسلة (الذي يجب أن يكون نفسه لكل السلاسل كونها موضوعة في مصفوفة) هو البعد الثاني. يبين الشكل (7-9) كيف يبدو هذا.



الشكل (9-7) مصفوفة سلاسل فارغة

نستطيع كتابة التعليمات التالية لإدخال بعض الأسماء في هذه المصفوفة:

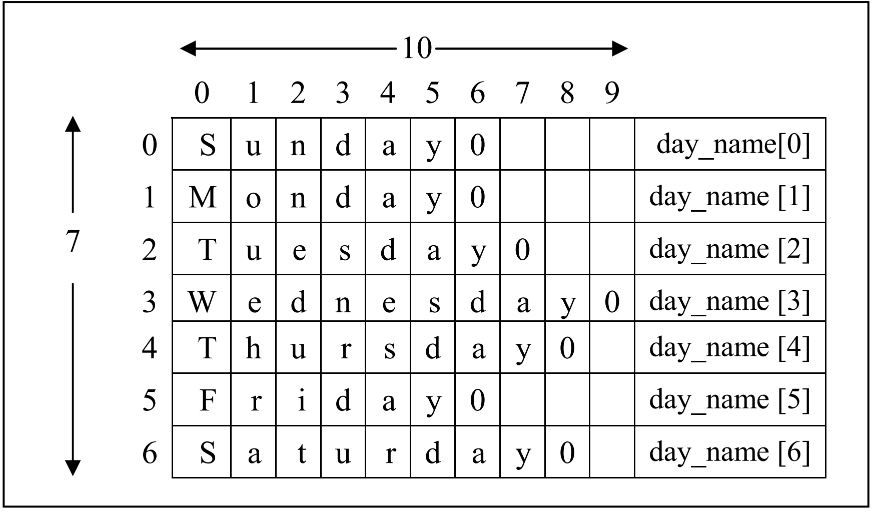
```
for (int j=0; j<5; j++)
    {
        cout <<"enter name (or press enter to exit loop):";
        cin.getline(names[j],10);
        if (strlen(names[j]) = = 0)
           break;
    }
```

لن تتيح الحلقة for للمستخدم إدخال أكثر من خمسة أسماء. وبضغط Enter وبالتالي إدخال اسم طوله صفر في المصفوفة الموافقة. يستطيع المستخدم إنهاء الحلقة بعد إدخال أقل من خمسة أسماء.

يمكن تمهيد قيم ابتدائية لمصفوفة سلاسل عند التصريح عنها. مثال:

```
const int max = 10;                 //  maximum length of day name,+1
const int dpw = 7;                  //  days per week
const char day_name[dpw][max]={"Sunday","Monday","Tuesday",
                               "Wednesday", "Thursday", "Friday",
                               "Saturday"};
```

كطريقة إعطاء قيم أولية لمصفوفة عددية نفسها، يفصل ما بين العناصر الفاصلة العادية. و توضع قيمة كل عنصر ما بين أداتي الحصر العلويتين لأن كل منها مصفوفة محرفية . يمكننا تمثيل ذلك كما في الشكل (9-8) التالي:



	0	1	2	3	4	5	6	7	8	9	
0	S	u	n	d	a	y	0				day_name[0]
1	M	o	n	d	a	y	0				day_name [1]
2	T	u	e	s	d	a	y	0			day_name [2]
3	W	e	d	n	e	s	d	a	y	0	day_name [3]
4	T	h	u	r	s	d	a	y	0		day_name [4]
5	F	r	i	d	a	y	0				day_name [5]
6	S	a	t	u	r	d	a	y	0		day_name [6]

الشكل (9-8) المصفوفة السلسلية التي تمثل أيام الأسبوع

لقد صرحنا عن بعدي المصفوفة max و dpw كمتغيرات ثابتة وكذلك صرحنا عن المصفوفة day_name بأنها من النوع const لأن أيام الأسبوع ثابتة لاتتغير.

من الشكل (9-8) نلاحظ أن هناك مساحات غير مستخدمة في نهاية كل سلسلة وذلك لأن أطوال السلاسل المكونة للمصفوفة السلسلية ليست متساوية. لذا يتم تبذير تلك المساحات ويمكن تجنب ذلك باستخدام المؤشرات.

سوف نعدل تعليمة التصريح عن مصفوفة السلاسل التي تعين أيام الأسبوع كعناصر لها، إلى مصفوفة مؤشرات إلى سلاسل كما يلي:

```
char*arrptrs [days]={"Sunday","Monday","Tuesday","Wednesday",
                     "Thursday","Friday","Saturday"};
```

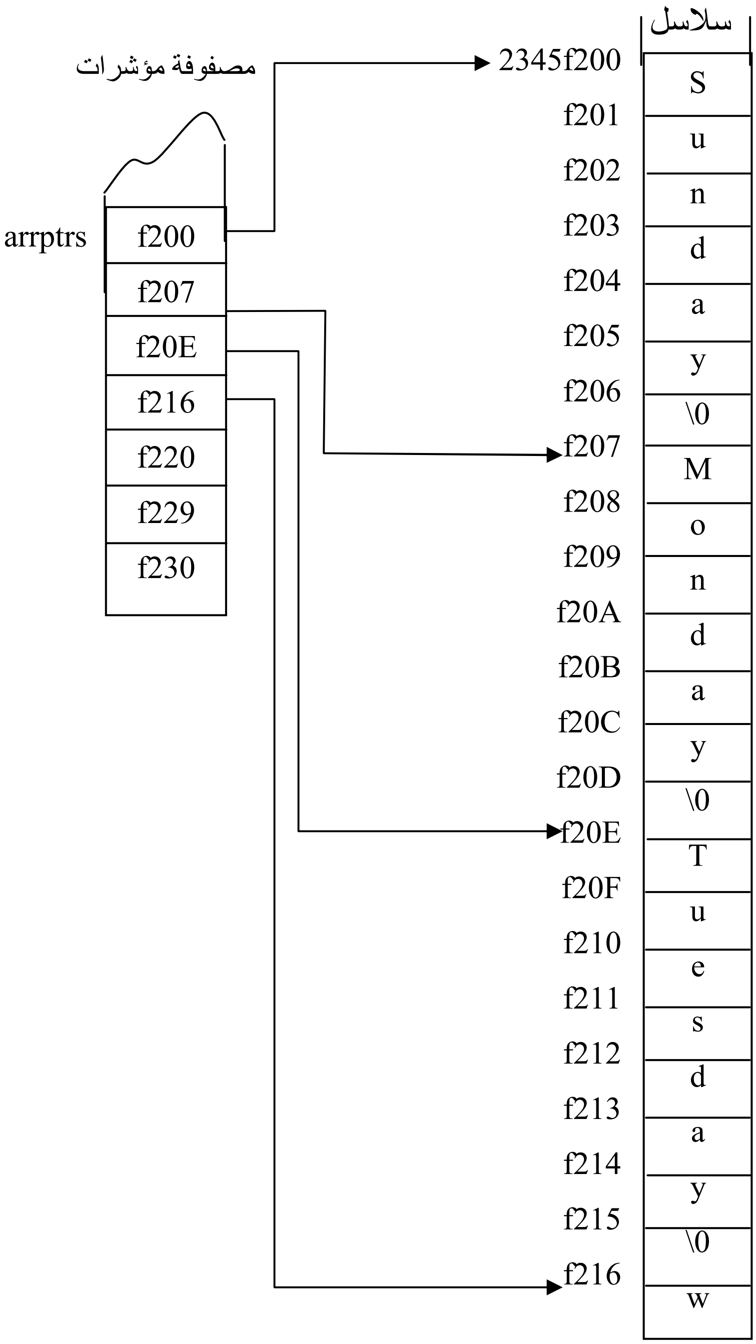
ويمكن طباعة هذه الأيام بالتعليمات التالية :

```
for(int j=0; j<days; j++)      // display every string
        cout << arrptrs [j] << "     ";
cout << endl;
```

والخرج يكون :

```
Sunday  Monday Tuesday  Wednesday Thursday  Friday  Saturday
```

عندما لاتكون السلاسل جزءاً من أية مصفوفة، تضعها c++ بشكل متجاور في الذاكرة لكي لايكون هناك أي تبذير. لكن يجب أن يكون هناك مصفوفة تخزن مؤشرات إلى السلاسل من أجل إيجادها. السلسلة نفسها هي مصفوفة من النوع char لذا مصفوفة المؤشرات إلى السلاسل هي مصفوفة مؤشرات إلى النوع char. هذا هو معنى التصريح arrptrs في التعليمة السابقة. تذكر أنه يتم تمثيل السلسلة دائماً بعنوان واحد هو عنوان الحرف الأول فيها. وهي العناوين التي يتم تخزينها في مصفوفة المؤشرات. الشكل (9-9) التالي يبين كيف يمثل ذلك.


مصفوفة مؤشرات
سلاسل
arrptrs
f200
f207
f20E
f216
f220
f229
f230
2345f200
f201
f202
f203
f204
f205
f206
f207
f208
f209
f20A
f20B
f20C
f20D
f20E
f20F
f210
f211
f212
f213
f214
f215
f216
S
u
n
d
a
y
\0
M
o
n
d
a
y
\0
T
u
e
s
d
a
y
\0
w

الشكل (9-9) مصفوفة مؤشرات إلى سلاسل

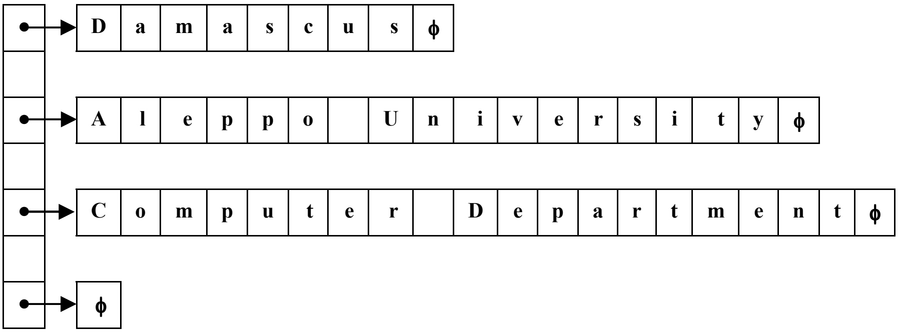
مثال (32) : البرنامج التالي يقرأ تتابعاً من الأسماء، اسم لكل سطر ينتهي الإدخال بطباعة الحرف ' $ ' . ثم يطبع الأسماء المخزنة في مصفوفة مؤشرات name.

```
# include < iostream.h >
main ()
{
        char array [80];
        cout << " enter names " << endl;
        cin.getline (array, 80, '$');
        char * name [4];
        name [0] = array;
        int count = 0;
        for (char * p = array; * p! = '\0'; p++)
            if (*p = = '\n')
            { *p = '\0';          // end name [count]
                name[++count] = p+1;       // begin next name
            }
        cout << " The names are: \n";
        for (int i=0; i<count; i++)
            cout << "\t" << i+1 << ": ["<<name[i]<<"]"<<endl;
        return 0;
}
```

وخرج البرنامج هو :

```
enter names
Damascus↵
Aleppo University↵
Computer Department↵
$↵
The names are:
        1: [Damascus]
        2: [Aleppo University]
        3: [Computer Department]
```

الحلقة for تفحص الدخل array باستخدام المؤشر p. كل مرة يجد المؤشر p الحرف '\n' ينهي السلسلة في name[count] بإلحاق الحرف '\0' إليها، ثم تزيد العداد count وتخزن العنوان p+1 للحرف التالي في name[count] وتكون المصفوفة الناتجة مشابهة للشكل (9-10) التالي:


D a m a s c u s ϕ
A l e p p o U n i v e r s i t y ϕ
C o m p u t e r D e p a r t m e n t ϕ
ϕ

الشكل (9-10) تمثيل مصفوفة مؤشرات

لاحظ أن البايتات التي أضيفت في نهاية أسماء أيام الأسبوع في الشكل (9-8)ليست مطلوبة مع المؤشرات.

مثال (33) : البرنامج التالي يبدأ مصفوفة المؤشرات بأسماء في تعليمة التصريح ثم يطبع محتوياتها.

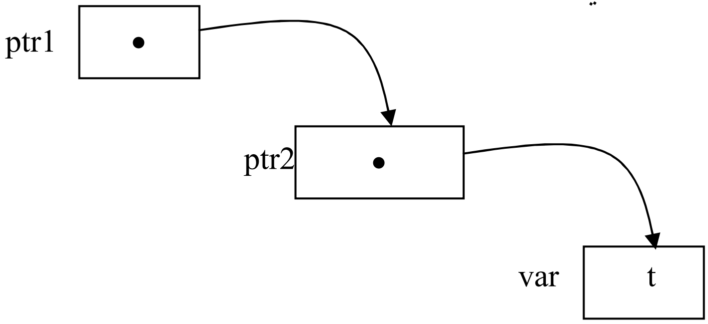
```
# include < iostream.h >
main ()
{
        char*names[ ] = { " Damascus","Aleppo University",
                                "Computer Department"};
        cout << " The names are : \n";
        for (int i=0; i<3; i++)
            cout <<"\t"<< i+1 << ":["<< names[i] << "]" << endl;
        return 0;
}
```

والخرج هو :

```
The names are:
            1:[ Damascus]
            2: [Aleppo University]
            3: [Computer Department]
```

10-2-9- مؤشرات لمؤشرات (Pointer to pointer):

في حالة المؤشر العادي، تكون قيمة المؤشر هي عنوان الذاكرة للمتغير الذي يحوي القيمة التي يشير إليها. بينما في حالة المؤشر إلى مؤشر، فإن المؤشر الأول يحوي عنوان المؤشر الثاني الذي بدوره يحوي عنوان الذاكرة للمتغير الذي يحوي القيمة المرغوبة. ويمكن تمثيل ذلك كالتالي:



ويمكن كتابة التعليمات التالية:

```
char var = 't';
char*ptr2 = & var;
char**ptr1 = & ptr2;
** ptr1 = 'w';    // changes the value of var to 'w'
```

حيث تعين التعليمة الأولى القيمة t للمتغير var. أما التعليمة الثانية فتصرح عن مؤشر لقيمة محرفية وتشير إلى المتغير var، أما التعليمة الثالثة فتصرح عن مؤشر ptr1 لمؤشر ptr2 . ثم في التعليمة الأخيرة تعين القيمة w للمتغير الذي يشير إليه المؤشر ptr2 عن طريق تأشيرة على مؤشره ptr1.

إذن بنتيجة تنفيذ التعليمات السابقة تصبح قيمة var تساوي 'w'.

مثال (34): البرنامج التالي يعلن عن متغير صحيح var ومؤشر ptr1 ومؤشر آخر ptr2 ثم يطبع القيمة التي يشير إليها.

```
# include < iostream.h >
main ( )
{
        int var, * ptr1,**ptr2;
        var = 30;
        ptr1 = & var;
        ptr2 = & ptr1;
        cout << " var = " << var << " \t * ptr1 = " << * ptr1
              << " \t ** ptr2 = " << ** ptr2 << endl;
        ** ptr2 = 40;
        cout << " var = " << var << " \t *ptr1 = " << *ptr1
              << " \t ** ptr2 = " << ** ptr2 << endl;
        return 0;
}
```

والخرج هو :

```
        var = 30                *ptr1 = 30            ** ptr2 = 30
        var = 40                *ptr1 = 40            ** ptr2 = 40
```

9-2-11- التوابع التي ترجع مؤشرات (Functions that return pointers):

يتم التعامل مع التوابع التي تعود بمؤشرات ، تماماً كأنواع التوابع الأخرى. إلاّ أن مؤشرات المتغيرات لايمكن أن تكون صحيحة int ولا قيماً صحيحة بلا إشارة unsigned int ولكن عناوين لمواقع تخزين معينة في الذاكرة.

الشكل العام لتعليمة التصريح عن التابع الذي يرجع مؤشراً هو التالي:

Return type_specifier*Function_name(Parameters);

حيث:

Return type_specifier نوع القيمة المرجعة.

المؤثر * بعد النوع يدل على أن التابع يعود بمؤشر ، أي بعنوان موقع تخزين معين في الذاكرة.

Function_name: اسم التابع.

Parameters بارامیتر واحد أو أكثر تفصل بينها الفاصلة العادية.

من المفيد عند معالجة السلاسل الحرفية استخدام التوابع التي تعود بمؤشرات.

مثال (35) : البرنامج التالي يقوم بإدخال سلسلة محرفية ثم يستدعي التابع locateblank( ) الذي يبحث عن أول فراغ في السلسلة المدخلة وعندما يجده يرجع مؤشراً يدل على عنوان ذلك الفراغ ضمن السلسلة، و يطلب ادخال رمز لإستبداله، أما إذا لم يجد التابع فراغاً في السلسلة فإنه يرجع مؤشر Null. لذلك فإن نوع القيمة المرجعة لهذا ا لتابع هو char.

```
# include < iostream.h >
char*locateblank(char*);
main ( )
{
        char line[80];
        char*blank_ptr;
        char ch;
        cout << " \n enter a string : \n";
        cin.getline(line,80);
        blank_ptr=locateblank(line);
        if (blank_ptr = = NULL)
            cout <<" \n The string you entered does not
                            contain a blank.";
        else
        {
            cout << " \n enter a character to replace the blank:";
            cin >> ch;
            *blank_ptr = ch;
            cout<<"\n The new string is as follows"<<endl;
            cout << line;
         }
        cout << endl;
        return 0;
}
char*locateblank(char*ch_ptr)
{
        while (* ch_ptr!= '\0')
        {
              if (*ch_ptr = = '   ')
                  return ch_ptr;
              ++ch_ptr;
        }
        return NULL;
}
```

والخرج هو :

```
enter a string:
new home↵
enter a character to replace the blank:_
The new string is as follows
new_home
```

مثال (36): البرنامج التالي يدخل سلسلة محرفية ثم يستدعي تابعاً للبحث عن رمز معين ضمن السلسلة ويعود بمؤشر على مكانه إذا وجد رمزاً مساوياً له في السلسلة المدخلة.

```
# include < iostream.h >
char*pointer (char);
char array[30];
main ( )
{
        char symbol;
        symbol = 'H';
        cin.getline(array,30);
        cout<<"\n The symbol H exist at:"
            << pointer (symbol);
        return 0;
}
char * pointer (char a)
{
        int i = 0;
        while (( a! = array[i]) &&(array[i]! = '\0'))
            i++;
        return (& array[i]);
}
```

يعود التابع pointer بمؤشر عند تحقق الشرط الموجود داخل التابع أي مصادفة رمز مساوٍ لإحدى رموز السلسلة الحرفية array المدخلة والمساوية لـ Say Hallo to your friend ، والخرج هو :

```
Say Hallo to your friend↵
The symbol H exist at: Hallo to your friend.
```

9-2-12- المؤشرات إلى توابع (Pointer to function):

التابع كالمتحول يحجز مكاناً في الذاكرة وقد يعين عنوان هذا المكان إلى مؤشر. وعنوان التابع يحدد بنقطة الدخول إليه ولهذا يمكن استخدام المؤشر إلى تابع لاستدعاء التابع.

تستخدم المؤشرات للتابع بنقل تنفيذ البرنامج إلى التابع المناسب حسب الخيار المعطى من قائمة من الخيارات وهي طريقة أسرع من استخدام تعليمة switch .

يعلن عن مؤشر لتابع بالشكل العام التالي:

```
return_type (*pointer_name) ( parameters_list)
```

لاحظ وجود قوسين حول pointer_name *لأن العملية * لها أولوية أضعف من تلك المتعلقة بالقوسين المحيطين بقائمة أنماط البارامترات.

حيث :

return_type نوع القيمة التي يعيدها التابع.

pointer_name اسم المؤشر إلى التابع

parameters_list أنواع بارامترات التابع.

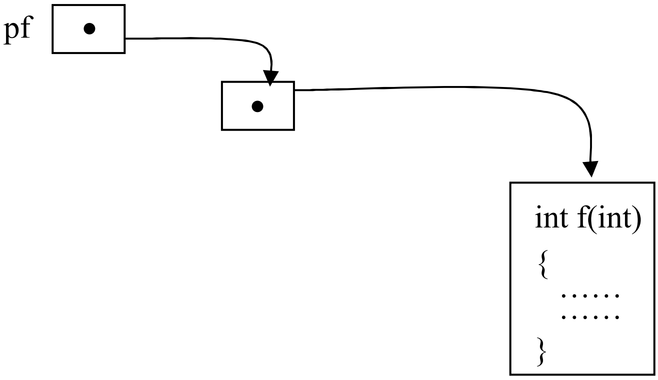
مثال :

```
int (* com) (int, int);
```

يعلن com مؤشر لتابع يرجع قيمة صحيحة وله متغيران من النوع الصحيح ، وإذا لم نضع القوسين حول com * فإن التصريح السابق يصبح int*com(int,int) وهو يعبر عن تابع اسمه com وله بارامتران من نوع int ويعيد قيمة هي عبارة عن مؤشر إلى int.

```
int f(int);      // declares the function f
int (*pf) (int);    // declares the function pointer pf
pf = &f;       // assign the address of f to pf
```

يمكن أن تتصور مؤشر التابع كالتالي:



على اعتبار أن اسم التابع مثل اسم المصفوفة هو في الحقيقة مؤشر ثابت، قيمته هي عنوان الشيفرة التي تنفذ التابع. والمؤشر لتابع هو ببساطة مؤشر قيمته عنوان اسم

التابع. حيث إن هذا الاسم هو نفسه مؤشر، لذلك فإن المؤشر لتابع هو مجرد مؤشر لمؤشر ثابت.

مثال (37): البرنامج التالي يستدعي تابعاً sum( ) له باراميتران، الأول هو مؤشر لتابع pf والثاني عدد صحيح n . التابع sum يحسب التابع الذي يشير إليه pf في كل من الأرقام الصحيحة 1 إلى n ويعيد مجموع هذه القيم التي عددها n. الإعلان عن باراميتر مؤشر التابع pf في قائمة باراميترات التابع sum( ) يحتاج إلى المتغير k.

```
# include < iostream.h >
int sum (int(*) (int), int);
int square (int);
int cube (int);
main ()
{
        cout << sum (square,4) << endl;  // 1+4+9+16
        cout << sum (cube,4) << endl;    // 1+8+27+64
        return 0;
}
// return the sum f(0)+f(1)+f(2)+…+f(n-1):
int sum (int(*pf)(int k),int n)
{
        int s = 0;
        for (int i=1; i<=n; i++)
            s + = (*pf) (i);
        return s;
}
int square (int k)
{
        return k*k;
}
int cube (int k)
```

```
{
       return k*k*k;
}
```

والخرج هو :

```
30
100
```

مثال (38) : البرنامج التالي يستدعي التابع fun الذي يوجد القيمة العظمى أو الصغرى بين عناصر مصفوفة array وذلك حسب قيمة أحد المتغيرات.

```
# include < iostream.h >
int max(int, int);
int min(int, int);
void fun (int*, int&, int(*)(int,int));
main ( )
{
        int num,ss;
        int array[ ] = { 2 , 1 , 7 , 4 , 5 };
        cout << " display array element : " << endl;
        for (int i=0; i<5; i++)
            cout << array[i] << "    ";
        cout << endl;
        cout << " enter 1 to fined the max, or 2 to fined the min:";
        cin >> num;
        if (num = =1)
        { fun (array, ss, max);
            cout << " the max is : " << ss << endl;
        }
        else if (num = =2)
                {
                fun(array,ss,min);
                cout << " the min is :" << ss<<endl;
                }
```

```
    return 0;
}
int max(int x,int y)
{
    if(x>y)
      return x;
    else
      return y;
}
int min (int x, int y)
{
    if(x<y)
      return x;
    else
      return y;
}
void fun (int*array, int&s, int(*pointer)(int,int))
{
    s=array[0];
    int i=0;
    while (i<5)
    {
      s = ((* pointer) (array[i], s));
      i+ = 1;
    }
}
```

والخرج هو :

```
display array elements:
2   1   7   4   5
enter 1 to fined max, or 2 to fined min:1↵
the max is: 7
```

لقد تم الإعلان في هذا البرنامج عن ثلاثة توابع:

التابع الأول max يرجع العدد الأكبر من بين اثنين مرسلين إليه.

التابع الثاني min يرجع العدد الأصغر من بين اثنين مرسلين إليه.

التابع الثالث fun يرجع العدد الأكبر أو الأصغر في مصفوفة array وذلك حسب القيمة المدخلة للمتغير num في البرنامج الرئيسي ( ) main.

يتضمن التابع fun مؤشر pointer يشير إلى كل من max و min حسب القيمة المدخلة للمتغير num. فإذا كان num مساوياً للعدد 1 أشار pointer إلى التابع max وأعاد المتغير s إلى المتغير ss القيمة العظمى في المصفوفة array . بينما لو كان num مساوياً للعدد 2 لأشار pointer إلى التابع min وأعاد القيمة الصغرى في المصفوفة array.

نؤكد على ضرورة كتابة المؤشر إلى التابع بالشكل:

int (* pointer) (int, int)

ليخبر التابع fun بتوقع وصول باراميتر على شكل مؤشر إلى تابع له باراميتران (وسيطان) من النوع int ويعيد قيمة من النوع int. وكذلك نؤكد على ضرورة وضع قوسين حول العبارة pointer* لأن العملية * لها أولوية أضعف من تلك المتعلقة بالقوسين المحيطين بقائمة أنواع البارامیترات . كما ذكرنا سابقاً.

13-2-9- مشاكل المؤشرات (Problems with pointers):

من الضروري استخدام مؤشرات أعطيت لها قيمة ابتدائية لأن مخالفة ذلك يسبب لنا كثيراً من المتاعب. وللمؤشرات كثير من الحسنات والمساوئ. فهي مثلاً تمدنا بطاقة عظيمة وهي ضرورية في كثير من البرامج إلاّ أن القيمة الخاطئة للمؤشر من الصعب جداً كشفها. وليس المؤشر بحد ذاته المشكلة وإنما ما ينتج عنه في كل مرة تنجز فيها عملية ما عليه.

فإذا كنا نقرأ منه فأسوأ ما يمكن أن يحدث هو أن نحصل على قيمة عديمة المعنى وخاطئة وعندما نكتب فقد نكتب فوق جزء آخر من برنامجنا أو معطياتنا، وهذا لايظهر إلاّ بعد تنفيذ البرنامج وقد يكون من الصعوبة معرفة مكان الخطأ. ومن أكثر الأخطاء الناجمة عن استخدام المؤشرات ما يلي:

1- الخطأ الناجم عن عدم تهيئة المؤشر بقيمة ابتدائية uninitialized pointer.

مثال (39) :

```
# include < iostream.h >
main ( )
{
        int a, *pointer;
        a = 10;
        * pointer = a;
        cout << * pointer << endl;
      return 0;
}
```

إن الإعلان int*pointer غير كاف فهو خصص ذكره للمؤشر pointer فقط ولكن قيمة المؤشر التي هي عنوان في الذاكرة لم تحدد بعد. هذا يعني أن عنوان الذاكرة هذا قد يكون مخصصاً بوساطة متغير آخر. لذلك عند تنفيذ البرنامج سيعطي المترجم تحذيراً بأن المؤشر استخدم بدون إعطاء قيمة ابتدائية. ولن ينفذ البرنامج حيث سيتم إعطاؤه رسالة مفادها أن البرنامج قام بعملية غير شرعية وسيتم إغلاقه . لتصحيح البرنامج لابد من كتابة التعليمة التالية بعد الإعلان عن المؤشر هي :

```
pointer = & a;
```

2- الخطأ الناجم عن عدم فهم كيفية استخدام المؤشرات. هذا موضح في البرنامج التالي.

مثال (40) :

```
# include < iostream.h >
main ( )
{
        int a, *pointer;
        a = 10;
        pointer = a;
        cout << * pointer << endl;
    return 0;
}
```

من الواضح أن المقصود في البرنامج هو طباعة قيمة a . لكن ذلك لن ينفذ لأن المترجم سيعطي خطأ وهو عدم إمكانية التحويل من النوع الصحيح إلى النوع مؤشر لنوع صحيح. يمكن للبرنامج أن ينفذ بتصحيح الخطأ وجعل تعليمة التعيين هي التالية:

```
pointer = & a;
```

وعندها يمكن طباعة قيمة a وهي 10.

9-3- تمارين

1- بفرض أن الأعداد الحقيقية وحيدة الدقة تخزن في الذاكرة في 4 بايت، وأن عنوان بداية المصفوفة في الذاكرة هو 1002500 ، وكل فقرة من هذا السؤال تعتمد على نتائج الفقرات السابقة لها. اكتب التعليمات التي تحقق ما يلي :

a- صرح عن مصفوفة حقيقية تدعى numbers ذات 10 عناصر، وعيّن القيم التالية لهذه العناصر : 0.0, 1.1, 2.2,..., 9.9 . إفرض أن القيمة الثابتة لـ size هي 10 .

b- صرح عن مؤشر nptr ليشير إلى قيمة حقيقية.

c- اطبع عناصر المصفوفة numbers باستخدام ترميز (اسم المصفوفة/دليل) في سطر واحد مع ترك فراغ بين كل عنصر وآخر.

d- اكتب تعليمتين تعينان عنوان بداية المصفوفة numbers للمؤشر nptr.

e- اطبع عناصر المصفوفة numbers باستخدام ترميز (مؤشر/إزاحة) باستخدام المؤشر nptr، وذلك في سطر واحد مع ترك فراغ بين كل عنصر وآخر.

f- اطبع عناصر المصفوفة numbers باستخدام ترميز (مؤشر/إزاحة) باستخدام اسم المصفوفة كمؤشر، وذلك في سطر واحد مع ترك فراغ بين كل عنصر وآخر.

g- اطبع عناصر المصفوفة numbers باستخدام ترميز (مؤشر/دليل) باستخدام المؤشر nptr، وذلك في سطر واحد مع ترك فراغ بين كل عنصر وآخر.

h- دل على قيمة العنصر (4) من المصفوفة numbers بأربعة طرق مختلفة.

i- بفرض أن nptr تشير إلى بداية المصفوفة numbers ، ما هو العنوان الذي تشير له العلاقة nptr+8 ؟ ما هي القيمة المخزنة في ذلك الموقع.

j- بفرض أن nptr تشير إلى numbers[5] ، ما هو العنوان الذي يشير له المؤشر nptr بعد تنفيذ التعليمة : nptr-=4 . وما هي القيمة المخزنة في ذلك الموقع.

```
a)  const int size = 10;
    float numbers [size] = {0.0, 1.1, 2.2, 3.3, 4.4, 5.5, 6.6, 7.7, 8.8, 9.9};
b)  float * nptr;
c)  for (int i=0; i<size; i++)
        cout <<  numbers [i] << "  ";
d)  nptr = numbers;
    nptr = & numbers [0];
e)  for (int i=0; i<size; i++)
        cout << * (nptr+i) << "   ";
f)  for (int i=0; i<size; i++)
        cout << * (numbers+i) << "   ";
g)  for (int i=0; i<size; i++)
        cout << nptr [i] << "   ";
h)  numbers [4];
    * (numbers+4);
    nptr [4]
    * (nptr + 4);
```

i) 1002532 = 4 * 8 + 1002500 : العنوان هو

والقيمة هي: 8.8

j) عنوان العنصر numbers [5] هو

1002500 + 4 * 5 = 1002520

عنوان القيمة: nptr-= 4 هو :

1002520 - 4 * 4 = 1002504

القيمة في ذلك الموقع هي : 1.1

2- أكمل الفراغات في البرنامج التالي الذي يقوم بجمع أو ضرب عددين باستخدام المؤشرات والتوابع الفرعية :

```
# include < iostream.h >
int add(….,….);
int multiple (….,….);
main ()
{
    int a , b;
    ……….;
    cout << " enter num1\n A= ";
    cin >> a;
    cout << " enter numb2\n B=";
    cin >> b;
    cout << " enter the operation \n";
    ……….;
    switch (x)
    {
       ……….;
       ……….;
       ……….;
       ……….;
       ……….;
       ……….;
    }
```

```
  return 0;
}
…… add (int * p, int * q)
     {
         ………..;
         ………..;
         cout << c << "\n";
         ………..;
     }
……. Multiple (int * p, int * q)
     {
         ………..;
         ………..;
         cout << " the results is" << c << "\n";
         ………..;
     }
```

البرنامج بعد ملء الفراغات:

```
# include < iostream.h >
int add(int*, int*);
int multiple (int* , int*);
main ()
{
     int a , b;
     char x;
     cout << " enter num1\n A= ";
     cin >> a;
     cout << " enter numb2\n B=";
     cin >> b;
     cout << " enter the operation \n";
     cin >>  x;
     switch (x)
     {
         case '+': add(&a,&b);
```

```
                    break;
          case '*': multiple (&a,&b);
                    break;
      }
   return 0;
}
int add (int*p , int*q)
      {
         int c;
         c=*p+*q;
         cout << c << "\n";
         return c;
      }
int multiple (int*p, int*q)
{
         int c;
         c = * p ** q;
         cout << " The result is" << c << "\n";
      return c;
}
```

3- أكمل الفراغات في كل من البرنامج وخرجه التاليين:

البرنامج :

```
# include < iostream.h >
int r, *k, & ref = … ;
main ()
{
      r =  400;
      k=&r;
      {
         int r, *k;
         r = …;
```

```
        k = & r;
        cout << " r equal : " << … << "\t" << " …="<<…<< endl;
    }
    {
        int r , *k;
        r = 100;
        k = & r;
        cout << " r equal: "<< r << "\t" = " << " ...."<<…<< endl;
    }
    cout << " r = " << r << " \t" << " ref = " << ref << endl;
    cout << " k = " << k << " \t" << " …" << *k << endl;
    return 0;
}
```

```
r equal : 200              * k = 200
………: 100                  k = 0x22a72410
…….=…….                   ………=………
…….=0x22a70e06             * k = …..
```

البرنامج بعد ملء الفراغات :

```
# include < iostream.h >
int r , *k , & ref = r;
main ()
{
    r = 400;
    k = & r;
    {
        int r, *k;
        r = 200;
        k = & r;
        cout << " r equal : " << r << "\t" << " *k= " << r << endl;
    }
    {
        int r, *k;
        r = 100;
        k = & r;
        cout << " r equal : " << r << "\t" << " k= " << k << endl;
    }
    cout << " r = " << r << "\t" << " ref = "<<ref << endl;
    cout << " k = " << k << "\t" << " *k = "<<*k << endl;
    return 0;
}
```

```
r equal : 200            * k = 200
r equal : 200            k = 0x22a72410
r = 400                  ref = 400
k = 0x22a70e06           * k = 400
```

4- اكتب برنامجاً يعين المؤشر s1 للقيمة الحرفية "Happy New Year" والمؤشر s2 لقيمة مساوية للقيمة السابقة، والمؤشر s3 لقيمة مساوية لـ "Happy Holiday"، ثم يقوم بطباعة ما يلي :

1- القيم الحرفية الثلاث كل في سطر منفصل.

2- ناتج مقارنة قيمة s1 مع s2 ، وكذلك ناتج مقارنة قيمة s1 مع s3.

3- ناتج مقارنة ستة رموز من s1 مع ستة رموز من s2 ، وكذلك ناتج مقارنة سبعة رموز من s1 مع سبعة رموز من s3.

4- ناتج نسخ رموز s3 إلى رموز s1.

5- ناتج حساب مجموع طولي s2 و s3.

```
# include < iostream.h >
# include < string.h >
main( )
{
    char * s1 = "Happy New Year";
    char * s2;
    char * s3 = "Happy Holiday";
    s2 = s1;
    cout << " s1 = " << s1 << endl << "s2 = " << s2
```

```
          << endl << " s3 = " << s3 << endl;
     cout << strcmp(s1,s2) << endl
           << strcmp(s1,s3) << endl;
     cout << strncmp (s1,s2,6) << endl
          << strncmp (s1,s3,7) << endl;
     cout << strcpy (s1,s3) << endl;
     cout << strlen (s2) + strlen (s3) << endl;
  return 0;
}
```

والخرج هو :

```
s1 = Happy New Year
s2 = Happy New Year
s3 = Happy Holiday
0
6
0
6
Happy Holiday
27
```

5- اكتب تعليمة واحدة لتنفيذ كل فقرة مما يلي :

a- التصريح عن السلاسل الحرفية التالية :

```
s1[50] = "Black"
s2[50] = "white"
s3[50]
```

وعن مؤشر لقيمة محرفية spr*

b- طباعة ناتج نسخ محارف s2 في s3 وفي سطر منفصل.

c- طباعة ناتج إضافة محارف s2 إلى ناتج إضافة "and" لناتج نسخ محارف s1 في s3 في سطر منفصل.

d- طباعة مجموع طولي السلسلتين الحرفيتين s1 و s2 في سطر منفصل .

e- طباعة طول السلسلة s3 في سطر منفصل.

```
a)  char s1[50] = "Black", s2[50] = "White", s3[50], * sptr;
b)  cout<< strcpy (s3,s2) << endl;
c)  cout<< strcat (strcat(strcpy(s3,s1),"and",s2)<< endl;
d)  cout<< strlen (s1) + strlen  (s2) << endl;
e)  cout<< strlen (s3) << endl;
```

وسيكون الخرج من تنفيذ التعليمات السابقة هو:

```
White
15
Black and white
12
```

6- اكتب برنامجاً لإدخال عشرة عناصر للمصفوفة الصحيحة A ثم يستدعي تابعاً فرعياً اسمه modify لمضاعفة قيم عناصر المصفوفة A ماعدا العنصر الخامس منها، وأخيراً يطبع قيم عناصر المصفوفة A بعد التعديل حسب ما يلي : في السطر الأول العناصر الأربعة الأولى ، في السطر الثاني العنصر الخامس، في السطر الثالث بقية العناصر.

```
# include < iostream.h >
void modify (int [ ], int);
main )
{
    const int arraysize = 10;
    int a [arraysize] , i;
    cout << " enter array a : " << endl;
    for (i = 0; i < arraysize; i++)
          cin >> a[i];
    cout << " print array elements : " << endl;
    for (i = 0; i < arraysize; i++)
          cout << a[i] << "    ";
    cout << endl;
    modify (a, arraysize);
    cout << " print array elements after calling function modify: "<<endl;
    for (i = 0; i < arraysize; i++)
       {
          if ( i! = 4)
               {
                    cout << a [i] << "    ";
                    continue;
               }
          cout << endl << a[i] << endl;
       }
    return 0;
}
    void modify (int b[ ], int size)
       {
          for(int i=0;i<size;i++)
            {
              if (i ! = 4)
              b[i] * = 2;
       }
}
```

والخرج هو :

```
enter array a:
1  2  3  4  5  6  7  8  9  10↵
print array elements:
1  2  3  4  5  6  7  8  9  10
print array elements after calling function modify:
2  4  6  8
5
12  14  16  18  20
```

7- اكتب برنامجاً يدخل قيمتين صحيحتين ويستدعي تابعاً فرعياً اسمه change يوجد جذر القيمة الأولى وعشرة أضعاف القيمة الثانية. ثم يطبع البرنامج الرئيسي القيمة الأولى وجذرها في سطر والقيمة الثانية وعشرة أضعافها في سطر آخر.

```
# include < iostream.h >
# include < math.h >
void change (int& , int&);
main()
{
    int a, b, c, d;
    cout << ' enter two integer values:";
    cin >> a >> b;
    c = a;
    d = b;
    change ( a,b );
    cout << " print a and its sqrt : " << endl;
    cout << " a = " << c << " \t sqrt (a) = " << a << endl;
    cout << " print b and 10*b : " << endl;
    cout << " b=" << d
          << " \t 10*b = " << b << endl;
    return 0;
}
void change (int & aref, int& bref)
    {
       aref = sqrt (aref);
       bref * = 10;
    }
```

والخرج هو :

```
enter two integer values : 9  3↵
print a and its sqrt:
a = 9              sqrt(a) = 3
print b and 10 * b:
b = 3              10 * b = 30
```

8- اكتب برنامجاً يدخل سلسلة محرفية ويطبعها كما أدخلت، ثم يطبعها بعد عكس أحرفها (أي الحرف الأول يصبح الحرف الأخير، والحرف الثاني يصبح الحرف ما قبل

الأخير). يقوم البرنامج باستدعاء تابع جزئي يدعى reverse يقوم بعكس أحرف تلك السلسلة. مرّر البارامترات للتابع باستخدام المؤشرات.

```
# include < iostream.h >
# include < string.h >
void reverse (char* , int);
main()
{
     char str [40];
     cout << " enter string str:";
     cin . getline (str, 40);
     cout << " The string str is : " << str << endl;
     reverse (str, strlen(str));
     cout << " The reversed string is : " << str << endl;
     return 0;
}
void reverse (char*s, int k)
{
     for (int i=0; i < k/2; i++)
     {
        char temp;
        temp = * (s+i);
        *(s+i) = * (s+k-i-1);
        *(s+k-i-1) = temp;
       }
}
```

والخرج هو :

```
enter string str : computer↵
the string str is : computer
the reversed string is : retupmoc
```

9- اكتب برنامجاً يدخل سلسلة محرفية ثم يستدعي التابع reverse الذي يعكس هذه السلسلة:

```
# include <iostream.h>
void reverse (char*);
main()
{
    char str [100];
    cout << "enter a string:";
    cin.getline (str, 100);
    cout << "the string is:"<< str<<endl;
    reverse (str);
    cout << " the reversed string is:"<< str<<endl;
  return 0;
}
void rverse (char*s)
{
    for (char*end = s; *end; end++); // find end of string
    while(s<end-1)
    {
    char temp = *--end;
       *end = *s;
          *s++=temp;
    }
}
```

والخرج هو:

```
enter a string: computer↵
the string is: computer
the reversed string is: retupmoc
```

10- نقول عن مصفوفة أنها متماثلة إذا كان نصفها الأيمن مطابق لنصفها الأيسر، مثال:

1 2 3 4 5 4 3 2 1

هي مصفوفة متماثلة. اكتب برنامجاً يستدعي التابعين enterarray و printarray لإدخال عناصر المصفوفة b وطباعتها. كما يستدعي التابع ispalindrame لإختبار المصفوفة

المدخلة هل هي متماثلة أم لا. يعيد التابع ispalindrame القيمة واحد إذا كانت المصفوفة الممررة له متماثلة، والقيمة صفر إذا لم تكن كذلك.

```
# include < iostream.h >
const int size = 20;
void enterarray (int*, int&);
void printarray (int*, int);
int ispalindrame (int*, int);
main()
{
    int b [size];
    int n, p;
    char ch;
    while (1)
    {
       cout << " enter array b :";
       enterarray (b,n);
       cout << " array b is :";
       printarray (b,n);
       p = ispalindrame (b,n);
       if (p = = 1)
           cout << " array b is palindrame."<< endl;
       else
           cout << " array b is not palindrame. " << endl;
       cout << " enter y to continue or another key to finish:";
       cin >> ch;
       if (ch ! = 'y')
           break;
       }
    return 0;
}
void enterarray (int* a, int & n)
{
      n = 0
      for (int*p = a; p < a + size; p++)
      {
         cin >> * p;
         if (* p = = 0)
             break;
             n++;
      }
}
void printarray (int * a, int n)
      {
         for (int * p = a; p < a + n; p++)
             cout << * p << "    ";
         cout << endl;
      }

int ispalindrame (int * a, int n)
{
      int m = 0;
      for (int i = 0, * p = a; i < n/2; i++)
         if(*(p+i) = = * (p+n-i-1))
             m++;
     if (m = = n/2)
         return 1;
      else
         return 0;
}
```

والخرج هو :

```
enter array b :   1   22   3   4   3   22   1  0↵
array b is :   1   22   3   4   3   22   1
array b is palindrame.
enter y to continue or another key to finish : y↵
enter array b :   1   2   3   4   3  0↵
```

```
array b is :   1   2   3   4   3
array b is not palindrome.
enter y to continue or another key to finish : n↵
```

11- اكتب برنامجاً يستدعي التابع enterarray لإدخال عناصر مصفوفة b، والتابع printarray لطباعة تلك العناصر مع عناوينها، والتابع maxaddress الذي يرجع مؤشر من النوع الصحيح قيمته عنوان القيمة العظمى بين عناصر المصفوفة المدخلة، والتابع minaddress الذي يرجع مؤشر من النوع الصحيح قيمته عنوان القيمة الصغرى بين عناصر المصفوفة المدخلة.

```
# include < iostream.h >
const int size = 20;
void enterarray (int *, int &);
void printarray (int *, int);
int * maxaddress (int *, int, int &);
int * minaddress (int *, int, int &);
main()
{
    int b [size];
    int m, max, min;
    int * p1, * p2;
    cout << " enter array b :";
    enterarray (b,m);
    cout << " array b is:" << endl;
    printarray (b,m);
    p1=maxaddress (b,m,max);
    p2=minaddress(b,m,min);
    cout << " the max = " << max << " was found at: "<< p1<<endl;
    cout <<" the min= "<< min<<" was found at:"<<p2<<endl;
    return 0;
}
void enterarray (int * a, int & k)
```

```
     {
        k = 0;
        for (int * p = a; p < a + size; p++)
            {
                cin >> * p;
                if (* p = = 0)
                     break;
                k++;
            }
     }
void printarray (int * a, int k)
     {
        int i = 0;
        for (int * p = a; p < a + k; p++)
            {
                cout<<"a["<<i<<"]="<<*p<<" is stored at:"<<p<<endl;
                i++;
            }
     }
int * maxaddress (int * a, int k, int & max)
     {
        int * p, * q;
        for (p = a, max = * p; p < a+k; p++)
            if (* p > max)
                {
                     max = * p;
                     q = p;     //     keep address of max
                }
     return q;
}
int * minaddress (int * a, int k, int & min)
     {
        int * a, * p;
        for (p = a, min = * p; p < a+k; p++)
            if (* p < min)
            {
                min = * p;
                q = p;      //   keep address of min
            }
        return q;
    }
```

والخرج هو :

```
enter array a : 8   13   6   22   5   20   4   0↵
array a is:
a[0] = 8 is stored at : 0x22e72500
a[1] = 13 is stored at : 0x22e72502
a[2] = 6 is stored at : 0x22e72504
a[3] = 22 is stored at : 0x22e72506
a[4] = 5 is stored at : 0x22e72508
a[5] = 20 is stored at : 0x22e7250a
a[6] = 4 is stored at : 0x22e7250c
the max = 22  was found at : 0x22e72506
 the min = 4     was found at : 0x22e7250c
```

12- اكتب برنامجاً يعين عشرة قيم صحيحة للمصفوفة a ثم يستدعي التابعين التاليين:

1) التابع swap يبادل بين قيمتين ممررتين له.

2) التابع ascending يرتب عناصر المصفوفة الممررة له تصاعدياً .

لترتيب عناصر المصفوفة تصاعدياً ثم يظهرها بعد الترتيب.

```
# include < iostream.h >
void swap (int*, int*);
void ascending (int*, int);
main()
{
    const int size = 10;
    int a [size] = {2, 4, 8, 6, 12, 10, 60, 30, 42, 88};
    cout << " data items in original order:" << endl;
    for (int i = 0; i < size; i++)
       cout << a{i] << "    ";
    cout << endl;
    ascending (a, size);
    cout << endl << " data items in ascending order:"<<endl;
    for (i = 0; i < size; i++)
       cout << a[i] << "    ";
    cout << endl;
  return 0;
}
void swap (int*n1, int*n2)
    {
       int temp;
       temp = * n1;
       * n1 = * n2;
       * n2 = temp;
    }
void ascendingint(int*a, int size)
{
    for (int pass = 1; pass < size; pass++)
       for (int i = 0; i < size-1, i++)
          if(a[i] > a [i+1])
              swap (& a[i], & a[i+1]);
}
```

والخرج هو:

```
Data items in original order:
2   4   8   6   12   10   60   30   42   88
Data items is ascending order:
2   4   6   8   10   12   30   42   60   88
```

13- اكتب برنامجاً يستدعي التوابع التالية بالشكل المناسب:

1) التابع ascending يعيد القيمة الصغرى بين قيمتين ممررتين له.

2) التابع descending يعيد القيمة العظمى بين قيمتين ممررتين له.

3) التابع swap يبادل بين قيمتين ممررتين له.

4) التابع bubble يسمح بترتيب عناصر مصفوفة تصاعدياً أو تنازلياً باستخدام مؤشر للتابعين ascending و descending.

وذلك لترتيب عناصر المصفوفة b تصاعدياً إذا أدخلت القيمة الحرفية 'v' أو تنازلياً إذا أدخل أي رمز آخر.

```
# include < iostream.h >
int ascending (int, int);
int descending (int, int);
void swap(int*,int*);
void bubble (int*, int, int (*) (int, int));
main()
{
    const int size = 10;
    char order;
    int b [size] = {2, 4, 8, 6, 12, 10, 60, 30, 42, 88};
    cout << " enter v to sort in ascending order," << endl;
    cout << " or another key to sort in descending order:"<<endl;
    cin >> order;
    cout << " data item in original order: " << endl;
    for (int i = 0; i < size; i++)
       cout << b[i] << "    ";
    if (order = = 'v')
       {
          bubble (b, size, ascending);
          cout << endl << " data items in ascending order:"endl;
       }
    else
```

```
        {
            bubble (b, size, descending);
            cout << endl << "data items in descending order:"endl;
        }
    for (i = 0; i < size; i++)
        cout << b [i] << "    ";
    cout << endl;
    return 0;
}
int ascending (int a, int b)
    {
        if(a > b)
            return b;
        else
            return 0;
    }
int descending (int a, int b)
    {
        if(a < b)
          return b;
    else
        return 0;
    }
void swap (int*n1, int*n2)
    {
        int temp;
        temp = * n1;
        * n1 = * n2;
        * n2 = temp;
    }
void bubble (int*b, int size, int(* compare) (int, int))
{
    for (int p = 1; p < size; p++)
```

```
        for (int count = 0; count < size-1, count++)
            if((*compare) (b[count], b[count+1]))
                swap (& b[count], &b [count+1]);
}
```

والخرج هو:

```
enter v to sort in ascending order,
or enter another key to sort in descending order:
v⏎
data items in original order:
2  4  8  6  12  10  60  30  42  88
data items is ascending order:
2  4  6  8  10  12  30  42  60  88
```

14- اكتب برنامجاً يسمح بإدخال عدد محدد من الأسماء ، اسم لكل سطر ثم يقوم بترتيب هذه الأسماء وإعادة طباعتها بعد ترتيبها. استخدم لذلك التوابع التالية:

1) التابع input يقبل بإدخال عدد من المحارف وتخزينها في buffer . ويتوقف عن الإدخال بقراءة الرمز $. حيث يتسع buffer (1+25*(1+20)) = 21*26 محرفاً.

2) التابع tokenize يقوم بتقسيم سلسلة buffer إلى سلاسل جزئية يفصل بينها محرف سطر جديد وتخزينها في المصفوفة name.

3) التابع print يقوم بطباعة الأسماء المخزنة في name.

4) التابع sort يقوم بترتيب الأسماء.

```
# include <iostream.h>
# include <strin.h>
const int name_length = 25;
const int max_num_names = 15;
const int buffer_length = (name_length+1)*max_num_names;
void input (char*);
void tokenize (char**, int&, char*);
void print (char**, int);
void sort (char**, int);
main()
{
     char* name [max_num_names];
     char buffer [buffer_length+1];
     int numNames;
     input (buffer);
     tokenize (name, numNarmes, buffer);
     print (name, numNames);
     sort (name, numNames);
     cout << " after sorting ";
     print (name, numNames);
   return 0;
}
//reads up to 15 strings into buffer:
void input (char*buffer)
{
     cout << "enter up to " << max_num_names<<"names,one per line."
           << "terminate with '$'. \n name are limited to"
           << name_length << "characters. \n";
     cint.getline (buffer, buffer_length, '$');
}
//copies address of each string in buffer into name array:
void tokenize (char**name, int&n, char*buffer)
{
     char*p = strtok (buffer, "\n");        //p points to each token
     for (n = 0; p&&*p! = '$'; n++)
     {
        name [n] = p;
        p = strtok (NULL, "\n");
     }
}
//prints the n name stored in buffer:
```

```
void print (char**name, int n)
{
    cout << "the names are: \n";
    for (int i = 0; i < n; i++)
       cout << "\t" << i+1<<":" << name[i] << endl;
}
//sort the n names stored in buffer:
void sort (char**name, int n)
{
    char*temp;
    for (int i = 1; i < n; i--)
       for (int j = 0; j < n- i; j++)
           if (strcmp (name[j], name[j+1]) > 0)
               {
                   temp = name[j];
                   name[j] = name[j+1];
                   name[j+1] = temp;
               }
}
```

والخرج هو :

```
enter up to 15 names, one per line. terminate with '$'.
name are limited to 25 characters.
ruba Sami↵
Ziad mahmod↵
ahmad ali↵
hanady ali↵
samar sami↵
$↵
the names are:
        1:  ruba sami
        2:  Ziad mahmod
        3:  ahmad ali
        4:  hanady ali
```

```
5:   samar sami
after sorting the names are:
1:   Ziad mahmod
2:   ahmad ali
3:   hanady ali
4:   ruba sami
5:   samar sami
```

15- اكتب برنامجاً يدخل محارف سلسلة str ثم يستدعي تابعاً فرعياً Length يوجد طول تلك السلسلة. التابع length يعيد طول السلسلة str حيث يمرر له مؤشر على بداية السلسلة str، ويمسح السلسلة حرفاً حرفاً ويخزن عدد المحارف في متغير محلي n.

```
# include <iostream.h>
int length (char*);
main()
{
    char str [40];
    cout << "enter string str:";
    cin.getline (str, 40);
    cout << "the length of the string str is:";
    cout << length (str) << endl;
  return 0;
}
int length (char * s)
{
    int n = 0;
    for (char * p = s; *p; p++)
          n++;
  return n;
}
```

والخرج هو :

```
enter string str: computer↵
```

```
the length of the string str is:8
```

16- اكتب برنامجاً يدخل قيم سلسلتين محرفيتين s2,s1 ثم يستدعي تابعاً فرعياً strcopy لنسخ محارف s2 فوق محارف s1 التابع strcopy يعمل عمل التابع strcpy ويقوم بنقل محارف s2 إلى s1 حتى يصادف المحرف الصفري للسلسلة s2 . لذلك لابدّ من نسخ المحرف الصفري إلى s1 بعد الخروج من عملية النسخ في الحلقة for.

```
# include <iostream.h>
char * strcopy (char*, char*);
main()
{
    char s1[50], s2[50];
    cout << "enter the first string s1:"<<endl;
    cin.getline (s1,50)
    cout << "enter the second string s2:"<<endl;
    cin.getline (s2,50)
    cout << "string s1 after calling strcopy is:"<<endl;
    cout << strcopy (s1,s2) << endl;
  return 0;
}
char * strcopy (char*s1, char*s2)
    {
       char*p1=s1;
       char*p2=s2;
       for (    ;*p2; p1++, p2++)
            *p1=*p2;
       *p1=NULL;
    return s1;
}
```

والخرج هو :

```
enter the first string s1:
computer↵
```

```
enter the second string s2:
technology⏎
string s1 after calling strcopy is:
technology
```

17- اكتب برنامجاً يدخل قيم سلسلتين محرفيتين s2,s1، ثم يستدعي تابعاً فرعياً strncopy لنسخ m محرف من s2 في بداية s1. يستمر النسخ من s2 إلى s1 طالما أن p2-52<m.

```
# include <iostream.h>
char * strncopy (char*, char*, int);
main()
{    char s1[50],s2[50]; int n;
     cout << " enter the first string s1:";
     cin.getline (s1,50);
     cout << " enter the second string s2:";
     cin.getline (s2,50);
     while(1)
     {
        cout << "enter n > 0, and 0 to end:";
        cin >> n;
        if ( n = = 0)
            break;
        cout << " string s1 after calling strncopy is: "<< endl;
        cout << strncopy (s1,s2,n) << endl;
     }
  return 0;
}
char * strncopy (char*s1, char*s2, int m)
     {
        char * p1=s1;
        char * p2 = s2;
        for (  ; p2-s2<m; p1++, p2++)
                *p1 = * p2;
        return s1;
```

```
    }
```

والخرج هو :

```
enter the first string s1: computer↵
enter the second string s2: technology↵
enter n > 0, and 0 to end : 4↵
the string s1 after calling strncopy is: techuter
enter n > 0, and 0 to end : 0↵
```

18- اكتب برنامجاً يدخل قيم سلسلتين محرفيتين s1 , s2 ، ثم يستدعي تابعاً فرعياً strconcat ينسخ محارف السلسلة s2 إلى نهاية محارف السلسلة s1 . التابع strconcat يعمل عمل التابع strcat . حلقة for الأولى تحدد نهاية السلسلة s1 . وبدءاً من هذا الموقع (مكان تخزين الحرف الصفري NULL للسلسلة الأولى) يبدأ نسخ السلسلة الثانية إلى السلسلة الأولى. تتوقف عملية النسخ عندما يشير المؤشر المحرفي p إلى NULL للسلسلة الثانية. لذلك لابدّ من نسخ الحرف الصفري إلى نهاية السلسلة s1 الجديدة والناتجة من نسخ s2 إلى نهاية s1.

```
# include <iostream.h>
char * strconcat (char*, char*);
main()
{
    char s1[50], s2[50];
    cout << " enter the first string s1:"<<endl;
    cin.getline (s1,50);
    cout << " enter the second string s2:"<<endl;
    cin.getline (s2,50);
    cout << " string s1 after calling strconcat is:"<<endl;
    cout << strconcat (s1,s2) << endl;
    return 0;
{
char * strconcat (char*s1, char*s2)
{
    for (char*end=s1; *end; end++); // find end of s1
    for (char*p=s2; *p; p++)
    {
       *end = *p;
       end++;
    }
    *end = '\0';
    return s1;
}
```

والخرج هو :

```
enter the first string s1:
computer↵
enter the second string s2:
technology↵
the string s1 after calling strconcat is:
computertechnology
```

19- اكتب برنامجاً يدخل قيم سلسلتين محرفيتين s2,s1 ، ثم يستدعي تابعاً فرعياً strnconcat ينسخ n محرفاً من s2 في نهاية السلسلة s1 . التابع str n con cat يعمل عمل التابع strncat. حلقة for الأولى تحدد نهاية السلسلة s1. وبدءاً من هذا الموقع (مكان تخزين NULL للسلسلة الأولى) يبدأ نسخ n محرف الأولى من السلسلة الثانية إلى السلسلة الأولى. تتوقف عملية النسخ عندما يصبح p-s2<n . لذلك لابدّ من إضافة المحرف الصفري إلى نهاية السلسلة s1 الجديدة.

```
# include <iostream.h>
Char*strnconcat(char*, char*, int);
main()
{
    char s1[50], s2[50];
    int n;
    cout << " enter the first string s1:"<<endl;
    cin.getline(s1,50);
    cout << "enter the second string s2:"<<endl;
    cin.getline(s2,50);
    cout<<"enter n > 0:";
    cint >> n;
    cout << " string s1 after calling strnconcat is:"<<endl;
    cout << strnconcat (s1, s2, n)<< endl;
  return 0;
}
char * strnconcat (char*s1, char*s2, int m)
{
    for (char * end = s1; *end; end++);  //  find end of s1
    for (char * p = s2; *p&& p-s2<m; p++)
       {
          *end = *p;
           end++;
       }
    * end = '\0';
  return s1;
}
```

والخرج هو :

```
enter the first string s1:
computer↵
enter the second string s2:
technology↵
enter n > 0:4↵
string s1 after calling strnconcat is:
computertech
```

20- اكتب برنامجاً يدخل جملة مكونة من عدة كلمات، ثم يقتطع كل كلمة من الجملة ويطبعها في سطر منفرد ، كما يقوم بحساب عدد الكلمات المؤلفة من أربعة حروف ويطبعه.

```
# include < iostream.h >
# include < string.h >
```

```
main()
{
     int k = 0;
     char * senptr , sen [ ];
     cin. getline (sen , 100 , '\n');
     cout << " the tokens are: " << endl;
     senptr = strtok (sen , "  ");
     while (senptr ! = NULL)
     {
        cout << senptr << endl;
        if (strlen (senptr) = = 4)
            k++;
        senptr = strtok (NULL, "  ");
     }
     cout << " number of words with four letters is equal to"
           << k << endl;
     return 0;
}
```

والخرج هو :

```
enter a sentence :
This is a book about food
the tokens are:
this
is
a
book
about
food
number of word with four letters is equal to: 3
```

21- اكتب برنامجاً يعين 10 قيم لعناصر المصفوفة الصحيحة a ، حيث تكون قيمة كل عنصر مساوية إلى قيمة دليلة مضروباً بالعدد 2 ، كما يدخل عدداً صحيحاً k ويستدعي تابعاً فرعياً للبحث عن القيمة k ضمن عناصر المصفوفة والعودة بموقع تلك القيمة إذا وجد قيمة مساوية لها وإلا العودة بالقيمة 1- . يقوم البرنامج بطباعة العبارة : " = The value k is found at I" ثم قيمة الموقع I وإلاّ يطبع العبارة :

" The value not found " إذا لم يجد قيمة مساوية لها.

```
# include <iostream.h>
int search (int [ ], int, int);
main()
{
    int a [10], k, b;
    cout << " enter the value k:";
    cin >> k;
    for (int i = 0; i < 10; i++)
       {
          a[i] = 2*i;
          cout << a[i] << "   ";
       }
    cout << endl;
    b = search (a, 10, k);
    if (b = = -1)
       cout << " the value not found " << endl;
    else
       cout << " the value k is found at I = " << b << endl;
    return 0;
}
int search (int a[ ], int c, int d)
    {
       int n = -1;
       for (int i = 0; i < c, i++)
```

```
            {
                if(a[i] = = d)
                n = i;
            }
    return n;
}
```

والخرج هو :

```
enter the value of k : 10↵
0   2   4   6   8   10   12   14   16   18
the value k is found at I = 5
```

22- اكتب برنامجاً يقوم بإدخال عشرة قيم صحيحة لمصفوفة أحادية b ثم يستدعي تابعاً add لجمع عناصر المصفوفة مستخدماً خاصية الاستدعاء المتكرر لنفس التابع. ومن ثم يطبع الناتج في سطر واحد بعد العبارة : "The result is" .

```
# include <iostream.h>
int add (int [ ], int);
main()
{
    const int arraysize = 10;
    int b [arraysize];
    cout << " enter array elements:" << endl;
    for (int i=0; i<arraysize; i++)
       cin >> b[i];
    int result = add (b, arraysize);
    cout << " The result is ="<< result << endl;
    return 0;
}
int add (int a[ ], int size)
    {
       if (size = = 1)
       return a[0];
       else
```

```
        return a[size-1] + add (a, size-1);
    }
```

والخرج هو:

```
enter array elements:
2  3  7  -2  8  10  15  20  11  9↵
The result is = 83
```

23- اكتب برنامجاً يقوم بإدخال عشرة قيم صحيحة لمصفوفة أحادية a ثم يستدعي تابعاً فرعياً اسمه Mull يقوم بإيجاد جداء عناصر المصفوفة a المدخلة باستخدام خاصية الاستدعاء المتكرر لنفس التابع ومن ثم يقوم البرنامج الرئيسي بطباعة الناتج في سطر بعد العبارة " :The result is ".

```
    # include < iostream.h>
long int Mull (int [ ], int);
    main()
    {
        int a[10];
        cout << " enter array element: " < endl;
        for (int i = 0; i < 10; i++)
            cin >> a[i];
        cout << " The result is: " << Mull (a,10) << endl;
        return 0;
    }
long int Mull (int b[ ], int k)
    {
        if (k = = 1)
            return b[0];
        else
            return b[k-1] * Mull (b,(k-1));
    }
```

والخرج هو :

```
enter array element:
```

```
1  2  3  4  5  2  3  4  5  6↵
The result is: 86400
```

24- اكتب برنامجاً يصرح عن مصفوفتين صحيحتين a[3][4] و b[3][4] ثم يستدعي التوابع التالية:

أ – التابع enterarray لإدخال عناصر مصفوفة ثنائية.

ب – التابع printarray لطباعة عناصر مصفوفة ثنائية على شكل أسطر وأعمدة.

ج – التابع comp يقوم بمقارنة عناصر مصفوفتين ثنائيتين لهما نفس الأبعاد وعدد العناصر، وتكوين مصفوفة ثالثة عناصرها 1 , 0 ,1- إذا كان العنصر المقابل في المصفوفة الأولى أكبر من أو يساوي أو أقل من العنصر المقابل في المصفوفة الثانية على الترتيب.

يتم استدعاء التوابع السابقة بالشكل المناسب لإدخال عناصر المصفوفتين b و a ومن ثم طباعتهما مع المصفوفة c الناتجة عن المقارنة.

```
# include <iostream.h>
# include <iomanip.h>
void enterarray (int [ ][4]);
void printarray (int [ ][4]);
void comp (int [ ][4]; int [ ][4], int[ ][4]);
main()
{
    int a[3][4], b[3][4]l, c[3][4];
    cout << " enter array a:";
    enterarray (a);
    cout << " enter array b:";
    enterarray (b);
    cout << " \n array a is:" << endl;
    printarray (a);
    cout << " \n array b is:" << endl;
    printarray (b);
```

```
     comp (a, b, c);
     cout << " \n array c is:" << endl;
     printarray (c);
     return 0;
}
void enterarray (int d[ ][4])
{
     for (int i=0; i<3; i++)
        for (int j=0; j<4; j++)
            cin >> d[i][j];
}
void printarray (int d[ ][4])
{
     for (int i=0; i<3; i++)
        {
            cout << setw (11);
            for (int j=0; j<4; j++)
                 cout << d[i][j] << setw(4);
            cout << endl;
        }
}
void comp (int a[ ][4], int b[ ][4], int c[ ][4])
{
     for (int i=0; i<3; i++)
        for (int j=0; j<4; j++)
            if (a[i][j] >  b[i][j])
                 c[i][j] = 1;
            else
                 if (a[i][j] < b [i][j])
                    c[i][j] = -1;
                 else
                      c[i][j] = 0;
}
```

والخرج هو:

```
enter array a: 1  2  3  4  5  6  7  8  9  10  11  12↵
enter array b: 2  1  3  5  4  6  6  7  9  8  12  11↵
array a is:
                1    2    3    4
                5    6    7    8
                9   10   11   12
array b is:
                 2    1    3    5
                 4    6    6    7
                 9    8   12   11
array c is:
                 -1    1    0   -1
                  1    0    1    1
                  0    1   -1    1
```

25- اكتب برنامجاً يدخل قيماً صحيحة لعناصر مصفوفة مربعة a ذات بعدين ويخرجها، ثم يستدعي تابعاً فرعياً يعين فيه قيماً صفرية لعناصر قطري تلك المصفوفة. ومن ثم يخرج البرنامج المصفوفة بقيمها الجديدة.

```
# include <iostream.h>
void arrzero (int [ ][3], int);
main()
{
    int a[3][3];
    cout << " enter array a: " << endl;
    for (int i=0; i<3; i++)
       {
          for (int j=0; j<3; j++)
             {
                cin >> a[i][j];
                cout << a[i][j] << "    ";
             }
          cout << endl;
       }
    arrzero (a,3);
    cout << endl << "print modified array a:"<< endl;
    for (i=0; j<3; i++)
       {
          for (j=0; j<3; j++)
              cout << a[i][j] << "    ";
          cout << endl;
       }
return 0;}
void arrzero (int b[ ][3], int c)
    {
       for (int i=0; i<c; i++)
          for (int k=0; k<c; k++)
              {
                  if((k = = i) || (k = = c-i-1))
                      b[i][k] = 0;
              }
    }
```

والخرج هو :

```
enter array a:
1   2   3   4   5   6   7   8   9↵
1   2   3
4   5   6
7   8   9
print modified array a:
0   2   0
4   0   6
0   8   0
```

26- استخدم خاصة التحميل الزائد للتوابع واكتب برنامجاً يقوم بإدخال زوج من المتغيرات ذات الأنواع التالية char,float,int ويستدعي التابع swap الذي يقوم

بتبديل قيمهما. يتوقف البرنامج عن التنفيذ بإدخال قيمتين متساويتين وتساويا الصفر.

```
# include < iostream.h >
 void Swap(int &,int &);
 void Swap(double & , double &);
 main()
{int n=12,m=54;
  double d=32.6,e=6.76;
cout<<"\n n="<<n<<"\t m="<<m<<endl;
cout<<"d ="<<d<<"\t e="<<e<<endl;
 Swap(n,m);
 Swap(d,e);
 cout<<"\n after swapping:"<<endl;
 cout<<"\n n="<<n<<"\t m="<<m<<endl;
 cout<<"d ="<<d<<"\t e="<<e<<endl;
 return 0;
 }
 void Swap(int & i,int & j)
   {int temp;
    temp=i;
    i=j;
    j=temp;
   }
 void Swap(double & x, double & y)
   { double temp;
      temp=x;
      x=y;
      y=temp;
   }
```

والخرج هو :

```
n=12                      m = 54
d = 32.6                  e = 6.76
after swapping
n = 54                    m = 12
d = 6.76                  e = 32.6
```

الفصل العاشر

أنواع البيانات المعرفة من قبل المستخدم

User defined data types

تسمح لغة ++c بتعريف أنـواع بيانـات خاصـة بـالمبرمج user-defined data types باستخدام أنواع البيانات التي تعرفنا عليها حتـى الآن، (مثـل char,float ...الخ). أهمها هي التي تستعمل الأصناف، كما سنوضح ذلك لاحقاً في الفـصل القـادم. ولكـن لنبدأ بالتعرف على تعليمة typedef و enum و struct.

10-1- تعليمة typedef :

تسمح هذه التعليمة بتعريف نوع بيانات معرف سابقاً باسم جديـد آخـر يختـاره المبرمج وتأخذ الشكل العام التالي:

typedef old_type_name new_type_name;

مثال (1):

typedef unsigned short int   USHORT;

هنا تم إعطاء اسم USHORT كاسم مرادف للنوع ذي الاسم unsigned short int.

مثال (2):

typedef char name[30];

حيث name هو اسم جديد للنوع char ويمثل مصفوفة طولها 30 رمزاً وبـذلك نستطيع استخدام هذا الاسم الجديد لتعريف متغيرات مـن النـوع نفـسه علـى النحـو التالي:

name first, mide, family;

تعليمة تصرح عن ثلاث مصفوفات محرفية كل منها بطول 30 رمزاً.

يتضح من الأمثلة السابقة أن استخدام الاسم البديل يساعد في اختصار التصريح الطويل كما في المثال (2). وكذلك في سهولة قراءة التصريح عن المتغيرات. كما في التعليمات التالية:

```
int minutes, hours;
typedef int Time;
Time minutes, hours;
```

التعليمة الأولى تصرح عن متغيرين من النوع int. التعليمة الثانية تعطي اسماً مرادفاً للنوع int وهو Time. التعليمة الثالثة تصرح عن متغير من نوع Time المكافئ للنوع int . الاستخدام الشائع لتعليمة typedef هو في إعطاء أسماء لنوع المراجع والمؤشرات.

مثال (3) : تعليمات التصريح التالية :

```
int* ptri;
double d=0;
double& dr=d;
```

يمكن أن تعوض بما يلي :

```
typedef int* pint;
typedef double& Rdouble;
pint ptri;
double d = 0;
Rdouble dr = d;
```

حيث تم استعمال أسماء النوع pint و Rdouble للتأكيد على نوع المرجع والمؤشر.

10-2- تعليمة نوع البيانات التعدادي (enumeration) :

يتيح نوع البيانات التعدادي – المحدد بالكلمة المفتاحية enum – للمبرمج أن يعرف نوع بيانات جديداً ومن ثم تحديد القيم المسموحة فيه. ويتم استعمال النوع

التعدادي عادة عندما يكون للمتغير بضع قيم مسموحة فقط، لأن كل قيمة يجب تسميتها.

الشكل العام للتعليمة:

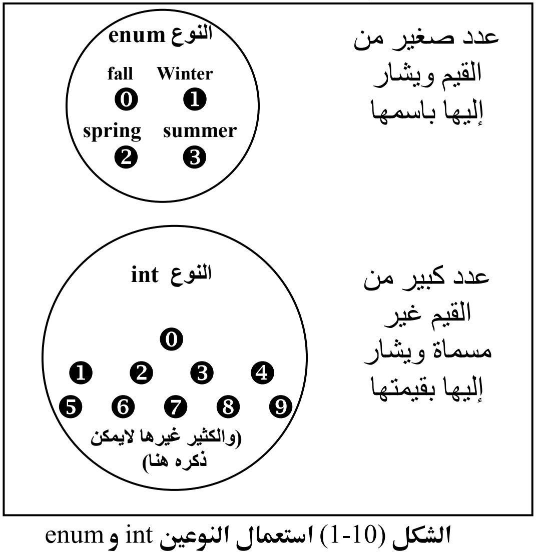
enum type-name { enumerator list };

مثال (4):

enum semester { fall,winter,spring,summer };

يلي الكلمة المفتاحية enum اسم النوع type-name وهو هنا semester ثم أقواس حاصرة { } تحيط بلائحة من أسماء قيم تفصلها فواصل وتنتهي التعليمة طبعاً بالفاصلة المنقوطة.

النوع التعدادي يعني أن كل القيم مذكورة قسرياً. هذا يختلف عن مواصفات متغير عدد، صحيح مثلاً، الذي له مجال من القيم الممكنة (32,768- إلى 32,767) . في enum علينا إعطاء اسم محدد لكل قيمة مسموحة. إن الأسماء المستعملة للقيم التعدادية ليست أسماء متغيرات. فالمتغير العددي الصحيح له قيمة 0 أو 1 أو 2 ...الخ. بينما المتغير التعدادي (حسب التعريف المبين أعلاه) فله قيم كـ fall و winter وصولاً إلى summer. الشكل (10-1) التالي يبين الفرق بين int و enum.


النوع enum
fall
Winter
⓿
❶
spring
summer
❷
❸
عدد صغير من القيم ويشار إليها باسمها
النوع int
⓿
❶ ❷ ❸ ❹
❺ ❻ ❼ ❽ ❾
(والكثير غيرها لايمكن ذكره هنا)
عدد كبير من القيم غير مسماة ويشار إليها بقيمتها

الشكل (10-1) **استعمال النوعين** int **و** enum

إن تحديد نوع تعدادي كما سبق لايؤدي في الواقع إلى إنشاء متغيرات تعدادية من ذلك النوع والإنشاء يتطلب تعليمة منفصلة.

10-2-1- إنشاء متغيرات تعدادية واستعمالها:

يمكن إنشاء متغيرات تعدادية وإعطاؤها قيماً كالتالي:

```
                         // specify a type enum.
enum days_of_week { Sun, Mon, Tue, Wed, Thu, Fri, Sat };
days_of_week day1, day2; // create variable of that type
day1 = Mon;       // give them values.
day2 = wed;
```

إحدى حسنات استعمال الأنواع التعدادية هي أن محاولة تعيين قيمة غير مذكورة ستؤدي إلى حدوث خطأ في الترجمة مما ينبهك إلى مكان العلة. مثلاً حتى ولو كان Easter تم تعريفه كمتغير عددي صحيح فإن التعليمة:

```
day1 = Easter;
```

لن يقبلها المترجم لأن day1 هو متغير من النوع days_of_week والكلمة Easter ليست مذكورة في لائحة القيم المحتملة لهذا النوع التعدادي.

في الواقع تخزن القيم التعدادية كأعداد صحيحة، بدءاً من 0 للقيمة الأولى عند الإعلان عن النوع. في الشكل (10-1) قيمة fall هي 0 و winter هي 1 و spring هي 2 و summer هي 3 . أما في نوع أيام الأسبوع فقيمة Sun هي 0 و Mon هي 1 و Tue هي 2 ... الخ. من ناحية أخرى يمكن تحديد بدء الترقيم بقيمة أخرى غير 0 ، أي يمكن إلغاء القيم التلقائية.

مثال :

```
enum grade { first = 1, second, third };
```

حيث تصبح قيمة first هي 1 و second هي 2 و third هي 3 . أي أنه لو خصصت قيم لبعض البنود من اللائحة، عندئذ فإن البنود التالية تأخذ قيماً متتابعة بزيادة واحد كل مرة. أو يمكنك تحديد قيم مختلفة لكل الأسماء مثال:

```
enum coin { penny = 1, nickle = 5, dime = 10, quarter = 25,
            halfdollar = 50, dollar = 100 };
```

يمكن أيضاً أن تتوافر بعض البنود بالقيمة نفسها، مثال:

```
enum Answer { no = 0, false = 0, yes = 1, true = 1, ok = 1 };
```

هذا يسمح بكتابة التعليمات التالية:

```
Answer ans;     // create variable ans of type enum
:
:
if (ans = = yes)
cout << "you said it was ok, \n";
```

فلو أن قيمة المتغير ans تساوي yes أو true أو ok (كل منهم مساوي 1) عند ذلك فإن الشرط المختبر صحيح ويتم تنفيذ طباعة العبارة على الشاشة.

تذكر أن القيمة الصحيحة 1 دائماً تعني true محقق في الشرط، لذلك بالإمكان كتابة التعليمة التالية:

```
if (ans)
cout << "you said it was ok. \n";
```

يمكن كتابة تعليمات التصريح التالية التي تصرح عن بيانات تعدادية مختلفة.

```
enum Boolean { false, true };
enum sex { femal, male };
enum Base { binary=2, Octal=8, decimal=10, hexadecimal=16 };
enum color { red, orange, yellow, green, blue, violt };
enum Rank { two, three, four, five, six, seven, eight, nine, ten,
            jack, queen, king, ace };
enum suit { clubs, dimonds, hearts, spades };
enum Roman { I=1, V=5, X=10, L=50, C=100, D=500, M=1000 };
```

التعريفات السابقة يمكن أن تجعل شيفرة البرنامج أكثر سهولة في القراءة . ولكن يجب عدم الإفراط في استخدامها. كل بند في القائمة يعرف مميزاً جديداً. على سبيل المثال التعريف Roman السابق يعرف أسماء مميزات I,V,X,L,C,D,M كثوابت

من الأعداد الصحيحة لذلك فإن هذه ا لحروف لايمكن استخدامها لأي غرض آخر داخل مجال تعريفهم.

أسماء البنود يجب أن تكون أسماء مقبولة. على سبيل المثال الأسماء التالية غير مقبولة:

```
enum grad { F, D, C-, C, C+, B-, B+, A-, A };
```

لأن الرموز '+' و '-' لايمكن استخدامها مع الأسماء.

10-3- البنية (structure) :

البنية هي نوع خاص من البيانات يحدد من قبل المبرمج. أو هي مجموعة من البيانات ذات الأنواع المختلفة والتي يمكن التعامل معها كوحدة واحدة من خلال اسمها أو مع عناصرها بشكل منفرد . بعكس المصفوفة المكونة من مجموعة من البيانات من النوع نفسه تحمل اسماً واحداً.

يتم استعمال البنية عندما تشكل عدة بيانات وحدة متميزة لكنها غير مهمة كفاية لتصبح صنفاً. البنى مهمة أكثر في لغة c حيث لاتوجد أصناف، أما في ++c فإن الصنف ينفذ العديد من المهام التي تقوم بها البنى في c . وبالرغم من ذلك مازال هناك كثير من الحالات التي تكون فيها البنى مفيدة.

10-3-1- تعريف البنية (Defining a structure)

يتم تعريف البنية في ++c كما يلي:

```
struct  structure_name
{
   variable_type1  variable_name1;
   variable_type2  variable_name2;
   ::::::::::::::::::::::::::::::::::::::::::::
   variable_typeN  variable_nameN;
};
```

حيث يبدأ تعريف البنية بالكلمة المفتاحية struct يليها اسم البنية وهو اسم مقبول في اللغة. ثم الأقواس الحاصرة { } التي تحيط بجسم البنية الذي يتألف من عدة بيانات يمكن أن تكون من أنواع مختلفة ، وتنتهي البنية بالفاصلة المنقوطة. البيانات المصرح عنها تدعى أعضاء البنية (struct member) ويتم التصريح عن هذه الأعضاء بالطريقة العادية التي يتم فيها التصريح عن المتغيرات. والتصريح عن البنية لاينتج عنه حجز مكان في ذاكرة الحاسوب وإنما يعني تحديد نوع يستخدمه البرنامج المترجم عند التصريح عن المتغيرات التي تتضمنها هذه البنية. أو إنه مجرد مواصفات لشكل البنية عندما يتم إنشاؤها.

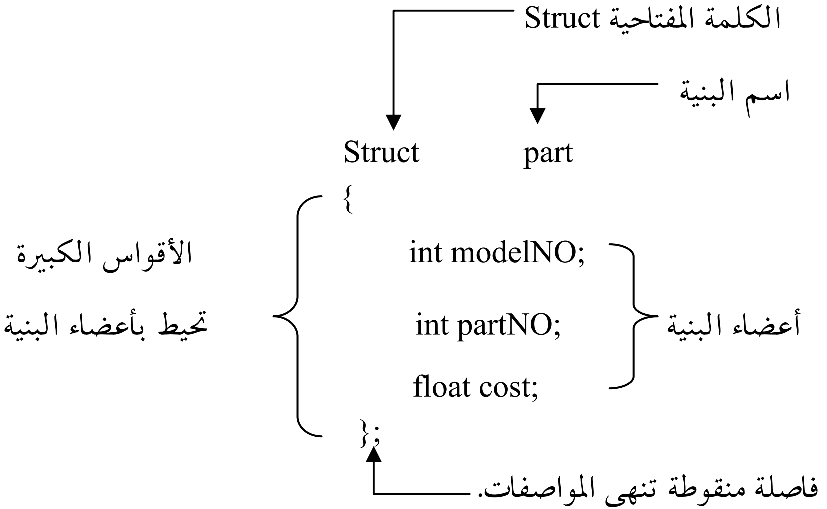
مثال (5): لنفرض أننا نود تخزين بعض البيانات عن ذاتية الموظفين كالاسم والعنوان والعمر والراتب. يمكننا فعل ذلك بالبنية المسماة employee التالية :

```
struct employee    //                      اسم البنية
{
   char name [ 30 ];  //  الحد الأدنى لعدد حروف الاسم هي 30
   char address [ 35 ];  //  الحد الأعلى لعدد رموز العنوان 35
   int age;
   float salary;
};
```

مثال (6): لنفرض أننا نود تخزين معلومات عن قطع غيار كرقم الموديل ورقم القطعة وثمنها. لأمكن ذلك بالبنية المسماة part التالية :

```
struct part
{
     int modelNo;
     int partNo;
     float cost;
};
```

يبين الشكل (2-10) تركيب البنية part



```
Struct        part
{
      int modelNO;
      int partNO;
      float cost;
};
```

الشكل (2-10) التركيب اللغوي للبنية

10-3-2- تعريف متغير من نوع البنية:

(Defining a structure's variable):

لكي نستخدم البنية المصرح عنها يجب تعريف متغير من نوع تلك البنية، وذلك كما يلي :

```
structure_name  variable_name;
```

مثال :

```
part cp1 , cp2;
```

المتغير cp1 أو cp2 يحجز في الذاكرة المكان اللازم لتخزين النوع المعرف سابقاً.

كما يمكن تعريف متغير من نوع بنية بالشكل التالي أيضاً :

```
struct part
{
        int modelNo;
        int partNo;
        float cost;
} cp1, cp2;
```

حيث يتم وضع cp1 و cp2 بعد القوس } ، وقبل الفاصلة المنقوطة.

10-3-3- الوصول إلى أعضاء البنية (Accessing structure members):

يتم استعمال عامل النقطة (٠) للوصول إلى أعضاء البنية. يأتي اسم المتغير من نوع البنية قبل النقطة بينما يأتي اسم العضو البياني بعده cp1.cost مثلاً :

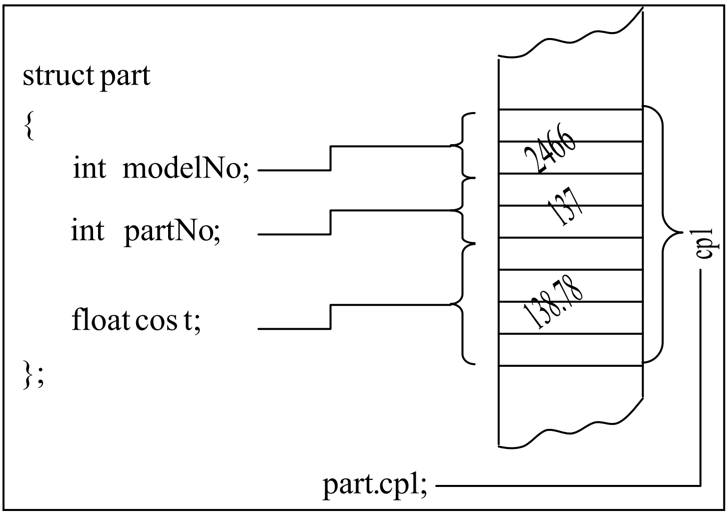
مثال (7): البرنامج التالي يعرف البنية part ويطلب إدخال قيم لأعضائها:

```
# include < iostream.h >
struct part       // specify structure
{
    int modelNO;
    int partNO;
    float cost;
};
main ( )
{
    part cp1;         // define structure variable
    cout << " Enter model number:";
    cin >> cp1. modelNO;        // Access data member in structure
    cout << " Enter part number:";
    cin >> cp1. partNO;      // Access data member in structure
    cout << " Enter cost :";
    cin >> cp1. cost;        // Access data member in structure
    cout << cp1. modelNO << "\t" << cp1. partNO << "\t"
         << cp1. cost << endl;
    return 0;
}
```

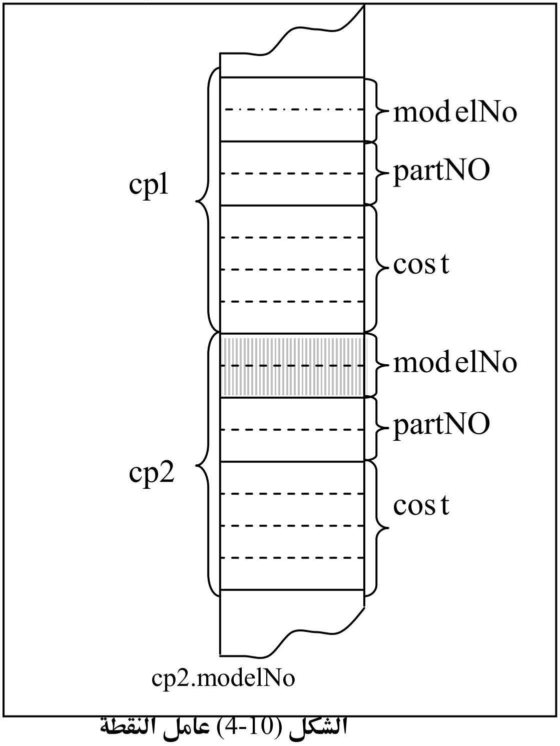
والخرج هو :

```
Enter model number : 2466↵
Enter part number : 137↵
Enter cost : 138.78↵
2466          137          138.78
```

يبين الشكل (10-3) كيف يبدو هذا التركيب إذا تم إدخال القيم 2466 و 137 و 138.78 كدخل للبرنامج. أما الشكل (10-4) فيعين طبيعة العلاقة بين اسم المتغير واسم العضو البياني وبين ما يتم تخزينه في الذاكرة.



الشكل (10-3) **أعضاء البنية في الذاكرة**



الشكل (10-4) **عامل النقطة**

10-3-4- إعطاء قيم ابتدائية لمتغير البنية (Initializing a structure's variable):

يمكن التصريح عن متغير بنية وإعطاؤه قيماً ابتدائية تماماً كما نفعل مع المصفوفات كما يلي :

```
part cp1 = { 2466, 137, 138.78 };
```

حيث عرف المتغير cp1 من نوع البنية part ومهد بقيم مثال (6) نفسها.

ينتج عن هذه التعليمة حجز مكان في الذاكرة وفقاً لنوع البنية part وتخزن القيم الثلاث الموجودة بين القوسين الكبيرين في المتغيرات الأعضاء لهذه البنية.

10-3-5- تعيين قيمة متغير بنية إلى متغير آخر (Assign a structure's variable value to another):

يمكن تعيين قيمة متغير بنية إلى متغير بنية آخر شريطة أن يكونا من النوع نفسه

تعليمة التصريح :

```
part  cp1 , cp2;
```

تصرح عن متغيرين من نوع البنية part المعرفة في المثال (6) في الفقرة (10-3-1) . وبفرض أعطيت القيم التالية لمتغير البنية cp1:

```
cp1 = { 2466 , 137 , 138.78 };
```

فإن نتيجة تنفيذ تعليمة التعيين التالية :

```
cp2 = cp1;
```

هي نسخ قيم المتغيرات الأعضاء للبنية cp1 على التتالي إلى المتغيرات الأعضاء المقابلة لها في البنية cp2.

10-3-6- مصفوفات البنية (Arrays of structure):

عرفنا فيما سبق أن المصفوفات لايمكن أن تحوي سوى نوع واحد من المتغيرات. ولكن باستخدام مصفوفات البنى يمكن أن تتضمن المصفوفة الواحدة عدة أنواع من المتغيرات.

في المثال (5) عرفنا بنية لذاتية موظف. لنفرض لدينا 100 موظف، عندها نستطيع كتابة تعليمات التصريح عن البنية أولاً ثم التصريح عن مصفوفة لذاتية الموظفين الـ 100 كما يلي :

```
struct employee
{
     char name[40];
     char address[40];
     int age;
     float salary;
};
employee emp[100];        // array of structure
```

ويمكن الوصول إلى عناصر المصفوفة كما يلي :

emp[3]. age = 30; //   عمر العنصر الرابع من المصفوفة

بالطريقة نفسها يمكننا تعريف بنية مصفوفات أي بنية بياناتها عبارة عن مصفوفات وذلك كما يلي:

```
struct array
{
     int arr1 [100];
     float arr2 [100];
     char str [100];
};
```

والتصريح عن متغير من نوع البنية array كما يلي :

```
array mat;
```

وللوصول إلى بيانات البنية mat نكتب كما يلي:

```
mat . arr1[0] , mat . arr2[0],
mat . arr1[1] , mat . arr2[1],
  :               :
  :               :
```

```
Mat . arr1[99] , mat . arr2[99],
```

البرنامج التالي يصرح عن متغير مصفوفة book[max] من نوع البنية bookInf التي تحوي بيانات عن مجموعة من الكتب.

مثال (8): اكتب برنامجاً لإدخال المعلومات التالية عن مجموعة من الكتب : عنوان الكتاب، اسم المؤلف، عدد الصفحات، سنة النشر، رقم الطبعة وذلك باستخدام البنية . ومن ثم طباعتها على الشاشة:

```
# include < iostream.h >
# include < string.h >     //     strcmp من أجل الإجراء

const int Max = 10;        //  عدد الكتب

struct   bookInf
{
     char title[40];          //   عنوان الكتاب

     char author[40];       //            المؤلف

     int pages;             //   عدد الصفحات

     int year;              //          سنة النشر

     char edition;           //            الطبعة

};
main( )
{
     bookInf book[Max];       //    مصفوفة الكتب

     int c , c1 , result;

     //  ابدأ باستقبال المعلومات من المستخدم

     for ( c = 0; c < Max; c++ )
     {
        cout << " Enter data for record NO: " << c+1;
        cout << " \t or enter \"quit\" to end \n\n";
        cout << " Enter title : ";
        cin >> book[c]. title;
        // إذا كتب المستخدم quit فإننا نخرج من التكرار
        result = strcmp (book[c]. title, "quit");
        if (result = = 0)
            break;
        cout << " Enter auther : ";
        cin >> book[c] . auther;
        cout << " Enter number of pages:";
        cin >> book[c] . pages;
        cout << " Enter year of publication:";
        cint >> book[c]. year;
        cout << " Enter edition:";
        cint >> book [c]. edition;
    }
        cout << "\n\n you entered:\n";
        //                اطبع المعلومات
        for ( c1 = 0; c1 < Max; c1++ )
        {
            cout << " Record No.\n" << c1+1;
            cout << " Book title:"<< book [c1]. title << endl;
            cout << " Book auther:"<< book [c1]. author << endl;
            cout << " No of pages:"<< book [c1]. pages << endl;
            cout << " Dat of pub:"<< book [c1]. year << endl;
            cout << " Book edition:"<< book [c1]. edition << endl;
        }
    return 0;
}
```

7-3-10- تمرير البنية إلى التوابع بالقيمة :

تسمح لغة c++ بتمرير البيانات المعرفة على شكل بنية إلى التوابع المعرفة مسبقاً باللغة أو التوابع المكتوبة من قبل المبرمج. يتم تمرير هذه البيانات بالقيمة ، حيث تمرر نسخة من البنية إلى التابع وبذلك لايمكن للتابع أن يعدل محتويات البنية الأساسية.

مثال (9) : اكتب برنامجاً لطلب معلومات عن أحد الطلاب وتخزينها في بنية من نوع studentinf ومن ثم تمريره إلى التابع print لتطبع محتوياته.

```
# include < iostream.h >
  struct studentinf
     {
        char name[30];
        int age;
        int grade;
     };
void print ( studentinf );
main ( )
{
     studentinf student;
     cout << " Enter student name:";
     cin >> student.name;
     cout << " Enter student age:";
     cin >> student. age;
     cout << " Enter student grade:";
     cin >> student.grade;
     cout << " you entered:\n ";
     print ( student );
     return 0;
}
void print ( studentinf st)
{
    cout << " student name : \t" << st.name << endl;
    cout << " student age:\t" << st.age << endl;
    cout << " student grade:\t" << st.grade<< endl;
}
```

والخرج هو :

```
Enter student name: Ali
Enter student age : 19
Enter student grade : 2
you entered
student name :    Ali
student age:        19
student grade:       2
```

ملاحظة : للبنية مدى وعمر المتغيرات العادية نفسه.

10-3-8- الحصول على بنية من تابع:

من الممكن أيضاً الحصول على قيمة من نوع بنية من استدعاء تابع ما.

مثال (10) : اكتب برنامجاً يصرح عن متغير من نوع بنية studentinf ويطلب من التابع getinfo إدخال المعلومات لهذا المتغير ثم استدعاء التابع print لطباعة تلك المعلومات.

```
# include < iostream.h >
struct studentinf
{
    char name [30];
    int age;
    int grade;
};
void print (studentinf);
studentinf getinfo (void);
main ( )
{
    studentinf student;
    student = getinfo ( );
    print (student);
    return 0;
}
void print (studentinf st)
{
    cout << " you entered : \n"
    cout << " student name: \t" << st.name << endl;
    cout << " student age: \t" << st. age << endl;
    cout << " student grade: \t" st. grade << endl;
}
studentinf getinfo (void)
{
    studentinf st;
    cout << " Enter student name:"
    cin >> st.name;
    cout << " Enter student age:"
    cin >> st.age;
    cout << " Enter student grade:"
    cin >> st.grade;
    return st;
}
```

والخرج هو :

```
enter student name: Ali
enter student age: 19
enter student grade: 3
you entered:
student name:    Ali
student age:     19
student grade:   3
```

مثال (11): البرنامج التالي يصرح عن بنية لتحديد الوقت ويصرح عن متغيرين من نوع تلك البنية ثم يستدعي تابعاً لتحديد أي من التوقيتين حدث أولاً ويطبعه.

```
# include < iostream.h >
struct time
{
     int hour;
     int minut;
     int second;
};
time min (time , time);
main ( )
{
     time t1 , t2 , t3;
     cout << " Time1.hour:";
     cin >> t1.hour;
     cout << " Time1.minut:";
     cin >> t1.minut;
     cout << " Time1.second:";
     cin >> t1.second;
     cout << " Time2.hour:";
     cin >> t2.hour;
     cout << " Time2.minut:";
     cin >> t2.minut;
     cout << " Time2.second:";
     cin >> t2.second;
     t3 = min (t1 , t2);
     cout << " hour:"<< t3.hour << endl;
     cout << " minut:"<< t3.minut << endl;
     cout << " second:"<< t3.second << endl;
     return 0;
}
```

```
time min ( time T1 , time T2)
{
   if ( T1.hour < T2.hour)
    return T1;
   else if ( T1.hour > T2.hour )
     return T2;
        else if ( T1.minut < T2.minut )
        return T1;
        else if ( T2.minut > T2.minut)
           return T2;
           else if ( T1.second < T2.second)
              return T1;
              else
                 return T2;
}
```

لقد تم تمرير نسخة من البنية إلى التابع وبذلك بقيت محتويات البنية الأساسية كما هي ونتيجة المقارنة هي بنية جديدة t3 مساوية للبنية التي تمثل بياناتها الزمن الذي حدث أولاً.

10-3-9- البنى المتداخلة:

يمكن لبنية أن تتضمن بنى أخرى. أي أن تكون المتغيرات الأعضاء في بنية ما هي أيضاً من نوع بنية أخرى.

فبفرض لدينا :

1- التصريح عن بنية تحمل الاسم تاريخ date وتتضمن بيانات أعضاء هي اليوم day والشهر month والسنة year.

2- التصريح عن بنية ثانية تحمل الاسم العنوان address وتتضمن بيانات أعضاء هي رقم الشارع number واسم الشارع street واسم المدينة city.

نستطيع استخدام تعريف البنيتين السابقتين date و address في التصريح عن بنية ثالثة تحمل اسم طالب student ولها بيانات الأعضاء وهي الاسم name والقسم department وهما من نوع محرفي أما العضو العنوان stuaddress فهو من نوع البنية address والعضو تاريخ الميلاد birthday من نوع البنية date

```
struct date
{
    char day;
    char month;
    int year;
};
struct address
{
    int number
    char stree[ ];
    char city [ ];
};
struct student
{
    char name [ ];
    char department;
    address stuaddress;
    date birthday;
};
```

بفرض أننا صرحنا عن متغير stu1 من نوع البنية student كما يلي :

```
student stu1;
```

عندئذ يمكننا الوصول إلى العضو الثاني من البنية student كما يلي:

```
stu1.department;
```

وإلى العضو الأول من العضو الثالث من البنية student كما يلي:

```
stu1.stuaddress.number;
```

مثال آخر : لتكن البنية education تحوي البيانات الأعضاء التالية: الاختصاص major والدرجة degree والمعدل average . وكذلك البنية students ذات البيانات الأعضاء التالية: المدرسة school من نوع البنية education والرقم التعريفي id وسنة التخرج ye-grade نستطيع كتابة ما يلي:

```
// The following structure has the members: major, degree, average.
struct education
{
    char * major;
    char degree;
    float average;
};
// The following structure is nested
struct students
{
    education school;
    char * id;
    int yr_grade;
};
```

نستطيع الآن إعطاء قيم ابتدائية لكل عضو من أعضاء البنية students والبنية education كما يلي:

1- تعريف اسم متغير من نوع البنية students

```
students Ali;
```

2- إعطاء قيم لأعضاء البنية الخارجية students كما يلي:

```
Ali . id = " 921345 ";
Ali . yr_grade = 2000;
```

3- إعطاء قيم لأعضاء البنية الداخلية school عن طريق المتغير Ali كما يلي:

```
Ali . school . major = " computer scince ";
Ali . school . degree = "B+";
Ali . school . average = 87;
```

مثال (12) : البرنامج التالي يعرف بنية لموظف employee تخزن فيها معلومات عن الاسم والعنوان والعمر والراتب. وبنية أخرى للقسم department تخزن فيها معلومات عن رقم القسم واسم المنتج ومتغير آخر من نوع البنية employee . ثم يصرح عن متغير من نوع البنية department ويطلب إدخال قيم لبيانات الأعضاء ومن ثم يطبعها على الشاشة.

```
# include < iostream.h >
struct employee
{
    char name [40];
    char address [40];
    int age;
    float salary;
};
struct department
{
    int dept-No;
    char product [40];
    employee emp;
};
main ( )
{
    department dept;
    cout << " Enter department number:";
    cin >> dept. dept_NO;
    cout << " Enter department product:";
    cin >> dept. product;
    cout << " Enter employee name:";
    cin >> dept. emp. name;
    cout << " Enter employee address:";
    cin >> dept. emp. address;
    cout << " Enter employee age:";
    cin >> dept. emp. age;
    cout << " Enter employee salary:\n";
    cin >> dept. emp. salary;
    cout << "  You entered:\n";
    cout << Department number:\t"<<dept.dept_No<<endl;
    cout << Department product:\t"<<dept.product<<endl;
    cout << Employee name:\t"<<dept.emp.name<<endl;
    cout << Employee address:\t"<<dept.emp.address<<endl;
    cout << Employee age:\t"<<dept.emp.age<<endl;
    cout << Employee salary:\t"<<dept.emp.salary<<endl;
    return 0;
}
```

والخرج هو :

```
Enter department number : 3
Enter department product : pen
Enter employee name : Ali
Enter employee address : Aleppo
Enter employee age : 32
Enter employee salary : 5432
You entered:
Department number:        3
Department product:       pen
Employee name:Ali
Employee address:   Aleppo
Employee age:   32
Employee salary:     5432
```

10-3-10- التصريح عن مؤشرات لبنية:

يتم التصريح عن مؤشرات لبنية بالطريقة نفسها التي يتم فيها التصريح عن المؤشرات للمتغيرات من الأنواع القياسية، وذلك كما يلي:

```
struct_name * pointer_variable_name;
```

مثال : بفرض لدينا البنية time التالية :

```
struct time
{
        int hour;
        int minute;
        int second;
};
```

يمكننا التصريح عن مؤشر لنوع البنية time كما يلي:

```
time * tptr;
```

للوصول إلى البيانات الأعضاء لمتغير المؤشر نستخدم العامل (<-) وذلك كما يلي :

```
tptr -> hour;
tptr -> minute;
tptr -> second;
```

بدلاً من النقطة (•) التي تستخدم للوصول إلى البيانات الأعضاء لمتغيرات من نوع time.

10-3-11- تمرير البنية إلى التوابع بالعناوين:

لقد ذكرنا سابقاً أن تمرير البارامترات بالعنوان إلى التابع تسمح للتابع بالتعامل والوصول إلى قيم البارامترات دون الحاجة لتكوين نسخة عنها في التابع. والمؤشرات هي إحدى طرق تمرير البارامترات إلى التوابع بالعنوان. البرنامج التالي يوضح ذلك.

مثال (13) : يمكننا إعادة كتابة البرنامج في المثال (11) باستخدام متغيرات مؤشرات وتمريرها إلى التابع:

```
# include < iostream.h >
struct time
{
    int hour;
    int minute;
    int second;
};
```

```
time * min (time*, time*);
main ( )
{
     time * tptr1,*tptr2,*tptr3;
     tptr1 = new time;
     tptr2 = new time;
     cout << " tptr1→hour:";
     cin >> tptr1→hour;
     cout << " tptr1→minut:";
     cin >> tptr1→minut;
     cout << " tptr1→second:";
     cin >> tptr1→second;
     cout << "tptr2→hour:"; cin >> tptr2→hour;
     cout << "tptr2→minut:"; cin >> tptr2→minut;
     cout << "tptr2→second:"; cin >> tptr2→second;
     tptr3 = min(tptr1,tptr2);
     cout << "tptr3→hour:"; << tptr3→hour << endl;
     cout << "tptr3→minut:"; << tptr3→minut << endl;
     cout << "tptr3→second:"; << tptr3→second << endl;
     return 0;
}
time * min (time*t1, time*t2)
{
     if (t1→hour < t2 → hour)
        return t1;
     else if (t1→hour > t2 → hour)
            return t2;
           else if (t1→ minut < t2 → minut)
                  return t1;
```

```
                    else if (t1 → minut > t2 → minut)
                              return t2;
                        else if (t1 → second < t2 → second)
                                  return t1;
                              else
                                return t2;
}
```

بهذه الطريقة يتم نقل عناوين السجلات في الذاكرة tptr1 و tptr2 كبارامترات إلى التابع min وليس بيانات هذه السجلات كما في المثال (11). القيمة المعادة هي أيضاً عنوان السجل الأصغر وليس بياناته.

10-4- تمارين

1- اكتب برنامجا يعين قيماً لمصفوفة أحادية مكونة من سبعة عناصر من النوع float تمثل درجة الحرارة العظمى لأيام الأسبوع السبعة. عرّف نوع تعدادي Day يمثل قيم أيام الأسبوع السبعة Sun ، Mon ، Tue ، Wed ، Thu ، Fri ، Sat . ثم يطبع الحرارة في كل يوم.

```
# include <iostream.h>
main()
{
     enum Day {Sun, Mon, Tue, Wed, Thu, Fri, Sat};
     float high [Sat+1] = {40, 39, 37, 37, 36, 36, 35};
     for (Day day = Sun; day < = Sat; day++)
          cout << " The high temperature for day " << day
                << "was" << high [day] << endl;
     return 0;
}
```

والخرج هو :

```
The high temperature for day 0 was 40
The high temperature for day 1 was 39
The high temperature for day 2 was 37
The high temperature for day 3 was 37
The high temperature for day 4 was 36
The high temperature for day 5 was 36
The high temperature for day 6 was 35
```

2- اكتب برنامجا يُعرف النوع التعدادي coins ويعين لها أنواع العملة التالية :

coins {Lera, two-lera, five-lera, ten-lera, twenty-five-lera};

ثم يصرح عن متغير من النوع التعدادي coins ويطبع عبارة مناسبة لقيمة ذلك المتغير.

```
# include <iostream.h>
main()
{
    enum coins {Lira, two-lira, five-lira, ten-lira, twenty-five-lira};
    money = two-lira;
    cout << "money=" << money << endl;
    if (money = = liera)
        cout << "you are broken"<<endl;
    cout << "you have to leave"<< endl;
    return 0;
}
```

الخرج هو :

```
money = 1
you have to leave
```

3- اكتب برنامجاً يعرف بنية Time لها ثلاثة متغيرات صحيحة hour و minute و second ، ثم يعرف متغير من البنية Time هو dintTime ويعطي القيم التالية لمتغيراته : hour = 18 , minute = 30 , second = 0 ومن ثم يطبع هذا الزمن إما بالشكل : 18:30:00 أو 6:30:00PM.

```
// create a structure, set its members, and print it.
# include <iostream.h>
```

```
struct Time       // structure definition
{
    int hour;              // structure definition
    int minute;            // 0-59
    int second;            // 0-59
};
void printform1 (const Time&);   // prototype
void printform2 (const Time&);   // prototype
main()
{
    Time dintime;
    // set members to valid values
    dintime. hour = 18;
    dintime. minute = 30;
    dintime. second = 0;
    cout << " Dinner will be held at:";
    printform1(dintime);
    cout<<"format 1 time"<<endl<<"which is  ";
    printform2 (dintime);
    cout<<"  standard time."<<endl;
    // set members to invalid values
    dintime. hour = 29;
    dintime. minute = 73;
    dintime. second = 103;
    cout << endl << "time with invalid values:";
    printform2(dintime);
    cout << endl;
    return 0;
}
// print the time in format 1
void printform1(const Time & t)
{
    cout << (t. hour < 10? "0" : "  ") << t. hour << ":"
```

```
          << (t. minute < 10? "0" : "  ") << t. minute << ":"
          << (t. second < 10? "0" : "  ") << t. second << ":" ;
}
// print the time in standard format
void printform2 (const Time & t )
{
     cout << (( t.hour = = 0 || t.hour = = 12 )? 12 : t.hour %12 )
           << ":" << (t.minute < 10? "0" : "  ") << t.minute
           << ":" << (t.second < 10? "0" : "  ") << t.second
           << (t.hour < 12 ? "AM" : "PM");
}
```

والخرج هو :

```
Dinner will be held at 18:30:00 format 1 time
which is 6:30:00 PM standard time.
time with invalid values: 5:73:103
```

4- عرّف بنية اسمها EMP_STRUCT تتكون مما يلي : اسم الموظف والرقم الذاتي ورقم الهاتف والعنوان بحيث يكون العنوان أيضاً عبارة عن بنية تتكون من اسم الشارع والمدينة والرمز البريدي ورقم المنزل.

5- تستخدم شركة لتأجير السيارات الجدول التالي لحساب الأجور المستحقة على الزبائن:

نوع السيارة	أجرة اليوم الواحد بالدولار	أجرة الكيلومتر الواحد بالدولار
مرسيدس	20	0.30
تويوتا	17	0.26
دايو	14	0.18
هونداي	12	0.15
سكودا	10	0.12

اكتب البرنامج اللازم لحساب الأجرة المستحقة باستخدام المعطيات في الجدول أعلاه.

يجب أن يطلب البرنامج من المستخدم إدخال نوع السيارة وعدد الأيام وعدد الكيلومترات المقطوعة خلال هذه الأيام. ويجب أن يبحث البرنامج في الجدول وإيجاد البيانات اللازمة وحساب الأجرة المستحقة (عن الأيام وعن الكيلومترات) ثم إظهار النتيجة على الشاشة. في حال عدم وجود نوع السيارة في الجدول يجب أن يظهر البرنامج رسالة تطلب من المستخدم تكرار عملية الإدخال.

6- اكتب برنامجاً يخزن بيانات الطلاب في صف ودرجاتهم الامتحانية لحساب المجموع وفقاً لما يلي :

1) تعريف بنية لتخزين البيانات التالية عن كل طالب : اسم الطالب ورقم الطالب ودرجات الامتحان الأول والثاني ودرجة المشاركة والامتحان النهائي ثم المجموع العام لهذه الامتحانات.

2) كتابة تابع يطلب من المستخدم إدخال البيانات المذكورة أعلاه، باستثناء المجموع لكل طالب من الطلاب، استخدم مؤشرات إلى البنية كباراميتر وحيد لهذا التابع.

3) كتابة تابع ذو بارامیتر عبارة عن مؤشر إلى البنية المعرفة في البند (1) يقوم بحساب المجموع النهائي للطالب.

يطلب البرنامج من المستخدم إدخال بيانات خمسون طالباً. ويخزنها في جدول، يستدعي البرنامج التوابع السابقة لتحقيق المطلوب وإظهار قائمة بأسماء الطلاب مرتبة أبجدياً وتبين المجموع العام لكل طالب.

الفصل الحادي عشر

الأصناف

Classes

11-1- برمجة الكيان (Object-Oriented Programming :O.O.P):

11-1-1- مقدمة :

لقد عانى المبرمجون من أساليب البرمجة التقليدية، فمن المعروف أن لغة الباسكال والبيزك والفورتران و c وغيرها من لغات البرمجة التقليدية هي لغات إجرائية (procedural). أي أن البرنامج المكتوب هو عبارة عن لائحة من التعليمات. وكل تعليمة في البرنامج تخبر الحاسوب أن ينفذ شيئاً، كجمع أرقام ، إدخال أخرى أو إظهار النتائج كخرج.

البرامج الصغيرة لاتحتاج عادة إلى أي مبدأ تنظيمي، فالمبرمج يضع لائحة التعليمات ويقوم الحاسوب بتنفيذها. أما في البرامج الكبيرة فإن وجود لائحة واحدة من التعليمات غير فعال، ولكي تصبح أكثر سهولة في الفهم، يتم تقسيمها عادة إلى أجزاء أصغر حجماً، ولهذا السبب تم اعتماد أسلوب التوابع، كطريقة لجعل البرامج أسهل للقراءة والفهم.

إن تقسيم البرنامج إلى توابع، حيث لكل تابع هدف معرف بشكل واضح، يمكن توسيعها بتجميع عدد من التوابع سوية في شيء أكبر يسمى وحدة (module). والمبدأ يبقى واحداً : أي مجموعة من التعليمات تنفذ مهاماً معينة. إن تقسيم البرنامج إلى توابع ووحدات هو أحد ركائز البرمجة البنيوية (Structured programming) . ولكن مع تزايد حجم البرامج وتعقيدها يبقى أسلوب البرمجة البنيوية ضعيفاً ويعود ذلك إلى عدة عوامل منها الاستخفاف بالبيانات (المعطيات) ، حيث إن أسلوب البرمجة البنيوية يركز على تنفيذ أشياء محددة كالقراءة من لوحة المفاتيح أو التحقق من وجود أخطاء أو ترتيب عناصر مصفوفة ... الخ .

ولكن ماذا يحصل للبيانات في هذا المنهج ؟ ... البيانات كما نعلم هي سبب وجود البرنامج، ومن المهم أن نحافظ عليها، ولكن البيانات المخزنة كمتغيرات عامة يمكن لأي تابع الوصول لها وتنفيذ عدة عمليات عليها وتغييرها. كما أن البيانات المعرفة كمتغيرات محلية لايمكن التعامل معها من خارج التابع المعرفة فيه. إن طريقة تخزين البيانات مهمة بسبب إمكانية عدة توابع الوصول إليها، ولو أردنا تغيير ترتيب البيانات لما أمكن ذلك من دون تغيير كل التوابع التي تستعملها. وعند إضافة بيانات جديدة لابدّ أيضاً من تعديل كافة التوابع التي تستعملها لكي تتمكن هذه التوابع من استعمال هذه البيانات الجديدة، وسيكون ذلك صعباً لأن علينا البحث عن كل التوابع وتعديلها جميعاً بطريقة متشابهة.

ولنشبه ذلك بما يحدث في متجر للتسوق (super_market) ، فلو أن إدارة المتجر نقلت مادة ما من الجناح رقم (i) إلى الجناح رقم (k). عندها يضطر المشتري إلى الاستعلام عن المكان الجديد للمادة وتغيير طريقة تبضعه المعتاد عليها وفقاً لذلك. مانحتاج إليه هو طريقة لحصر الوصول إلى البيانات، أي إخفاء البيانات عن التوابع ماعدا التوابع ذات العلاقة منها، سيؤدي هذا إلى حماية البيانات وتسهيل أعمال الصيانة والكثير غيرها من الفوائد.

من كل ذلك نخلص إلى أن الأهم في اللغات كائنية المنحى هو دمج البيانات والتوابع التي تعمل على تلك البيانات في كينونة واحدة تسمى كائن (object). وعند توافر عدة كائنات متطابقة في برنامج واحد فمن غير المنطقي وصف كل واحد منها على حدة، بل من الأفضل عندها تطوير مواصفات واحدة لكل هذه الكائنات مجتمعة. إن مواصفات إنشاء الكائنات في لغات O.O.P يدعى الصنف (class). وسندرس ذلك بالتفصيل ولكن لكي ندرك ونفهم ما هي برمجة الكيان، لابدّ من دراسة الصفات التي تتصف بها برمجة الكيان وهي:

11-1-2- الأصناف (التغليف) ((Classes (Encapsulation)):

يمكن وصف برمجة الكيان بأنها البرمجة المتعلقة بالبيانات (الصفات) والتوابع (أفعال behavior أو عمليات Operations) التي تستخدم تلك البيانات، حيث يتم

تسمية البيانات والتوابع باسم معين هو الكيان ويكون هذا الكيان مكتفياً ذاتياً (أي يشكل وحدة برمجية متكاملة).

تسمى التوابع الأعضاء أحياناً بالطرق (methods) في لغات البرمجة غرضية التوجه الأخرى ويتم التعامل معها بوساطة رسائل messages مرسلة إلى الكائن بحد ذاته . ترتبط كل رسالة بعملية استدعاء لأحد التوابع الأعضاء.

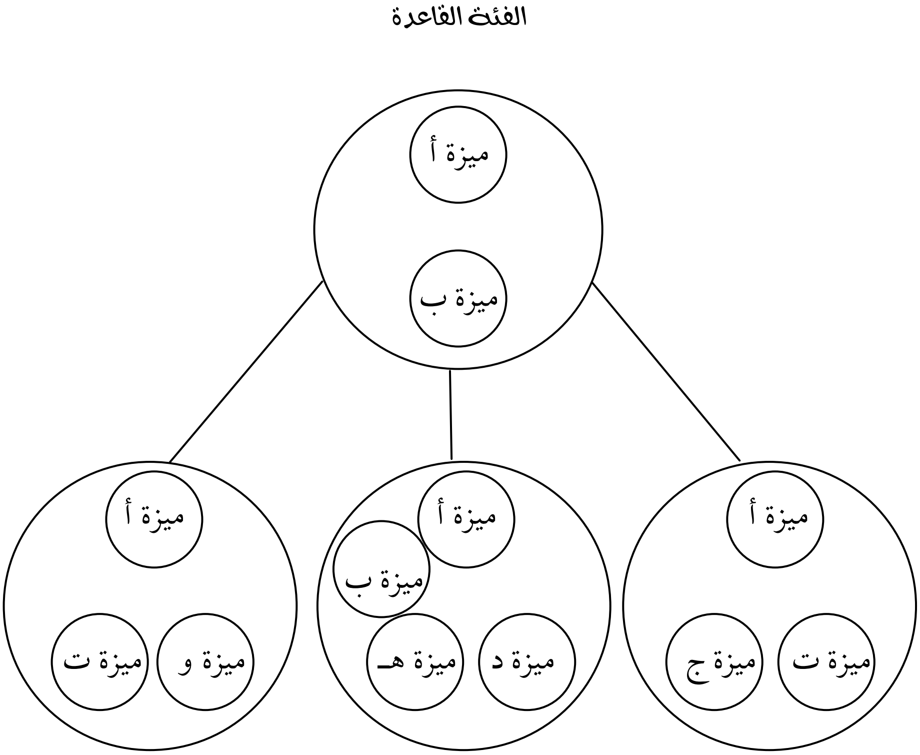
وحسب الوصف السابق فإن كثيراً من البرامج التي تقوم بمهمة معينة وتحتوي على البيانات التي تحتاجها لأداء عملها تسمى بالكيان. ويمكننا هنا التساؤل عن الطريقة التي توفرها لغة ++c وتؤدي إلى الكيان. الجواب هو عن طريق توفير ما يسمى بالأصناف (classes). وهذه الطريقة مشابهة لعملية تركيب البيانات (data structures)، ولكنها تتصف بصفات أخرى غير موجودة في عملية تركيب البيانات حيث تحتوي على برامج بحد ذاتها (تسمى التوابع أو العناصر التوابع)، وتعتمد هذه البرامج على تركيب البيانات، وبنفس طريقة تحديد حالة من حالات تركيب البيانات نستطيع تحديد حالة من حالات الصنف. وتحديد حالة صنف بحد ذاتها تسمى كياناً، أي وصف متغير ليعبر عن مواصفات الصنف نفسه.

توفر وسيلة الصنف (class) هذه مجموعة من الميزات في أداء البرمجة، حيث يمكن للمبرمج تعريف متغيرات تابعة للصنف ويطلق عليها متغيرات خاصة لايمكن الوصول إليها من خارج الصنف نفسه. وهذا يعني أن التوابع المعرفة لعناصر تنتمي للصنف نفسه تستطيع الإشارة أو استخدام المتغيرات المعرفة كمتغيرات خاصة. إن وصف قسم من المتغيرات على أنها متغيرات خاصة يساعد على حماية المعلومات من البرامج غير المسموح لها بالاستفادة من بيانات معينة.

إن استخدام طريقة الصنف يجعل البرامج الكبيرة واضحة المعالم حيث تكون جميع البرامج ذات العلاقة في مواقع محددة لها وهذا الأسلوب يؤدي إلى مفهوم الوحدات المتكاملة وهذا بالتأكيد يؤدي إلى الصيانة المجدية وتسهيلها.

10-1-3- التوارث (Inheritance) :

إن الصفة الثانية التي تتصف بها برمجة الكيان (غير صفة الصنف) هي التوارث أو الاستفادة من الخصائص المعرفة في برامج أخرى سابقة. وهذه الوسيلة تمكن المبرمج من الحصول على أصناف جديدة وذلك بالاستفادة من البرامج المكتوبة في أصناف أخرى سابقة بالإضافة إلى الخصائص الجديدة التي يمكن إضافتها وهذه الوسيلة توفر وقت المبرمج وجهده. وقد تساعد وسيلة التوارث على الحصول على معلومات من أكثر من صنف واحد وهذه تسمى بالتوارث المتعدد، أي الحصول على مساعدات من جهات مختلفة. الشكل (11-1) يوضح فكرة الوراثة بين الفئات:



الشكل (11-1) الوراثة بين الفئات المشتقة والفئة القاعدة

يوضح الشكل (11-1) أن كل فئة مشتقة فرعية تمتلك خصائص مشتركة مع الفئة التي تنحدر منها. الفئة الأساسية تمتلك الميزتين (أ) و (ب) وهما مشتركتان مع كل من الفئات المشتقة المنحدرة منها، لكن لكل فئة منحدرة خصائص أخرى خاصة بها.

في لغة ++C تسمى الفئة الأصلية الفئة القاعدة (base class) ، وبذلك يمكن تعريف فئات أخرى تتشارك معها خصائصها، لكن تضيف خصائصها الذاتية أيضاً. إنها تسمى الفئات المشتقة (derived classes) . الوراثة شبيهة نوعاً ما لاستعمال التوابع من أجل تسهيل البرامج الإجرائية التقليدية. لنفرض أن هناك ثلاثة أقسام مختلفة في برنامج إجرائي تقوم بالعمل نفسه تقريباً، يمكن عندها استخراج العناصر المشتركة من هذه الأقسام الثلاثة ووضعها في تابع واحد. وتستطيع الأقسام الثلاثة استدعاء هذا التابع لتنفيذ الأعمال المشتركة كما يمكنها تنفيذ أعمالها الخاصة أيضاً. ومثلما تفعل التوابع في البرنامج الإجرائي ، تقصر الوراثة البرنامج الكائني المنحى وتوضح العلاقة بين عناصره.

11-1-4- التوابع والأدوات المتعددة الأغراض (Polymorphism):

الصفة الثالثة التي تتصف بها برمجة الكيان والتي تؤدي دوراً مهماً، هي الاستعمال متعدد الأغراض للأدوات مثل + ، - ، * والتوابع. حيث يتم استخدام الأداة نفسها أو اسم التابع نفسه لأغراض متعددة، يمكن مثلاً، تعريف الأداة ' + ' لتمثل جمع عددين صحيحين أو حقيقيين أو حرفين، أو جمع اسمين (نصين). وبعبارة أخرى للتابع أو الأداة معانٍ متعددة حسب غرض الاستخدام. فمثلاً يمكن استخدام التابع وتمييزه من خلال نوع المتغيرات التي تمثله أو نوع النتائج وهذه يتم تمييزها، أو الغرض من الاستخدام خلال عملية الترجمة.

إن الصفات المذكورة أعلاه تجعل من لغة ++c لغة مثالية وذات قوة عالية حيث تعطي المبرمج حرية الاختبار وتقلل من حجم البرنامج المستخدم حيث يستطيع المبرمج

الاستفادة من كثير من التوابع والأصناف، وكما ذكرنا سابقاً يستطيع المبرمج تجهيز الهيكل العظمي الذي هو أساس البرنامج من خلال الأصناف المعرفة والمبرمجة سلفاً في برامج سابقة. وهذا التعرف يفيد في إعادة استخدام البرامج لأغراض متعددة مما يجعل من البرمجة وحدات متكاملة مشابهة للوحدات المادية للحاسب.

11-2- الأصناف (classes):

لغة ++C هي لغة موجهة. هذا يعني أن هذه اللغة جيدة في محاكاة المنظومات التي تتكون من كيانات متداخلة. مثل منظومة التحكم في المطار مثلاً. في مثل هذه المحاكاة، الكيانات في المنظومة (الطائرات، الناس، الحقائب، ...الخ) . تكون ممثلة بمتغيرات في برنامج الحاسوب. لذلك فإن المتغيرات ينظر إليها على أنها الكيانات نفسها. ويمكن رؤيتها كوحدات قائمة بذاتها لها قدرات خاصة.

لقد تعلمنا كيف نخلق كياناً من أنواع بيانات مثل char, int ، ...الخ. وذلك باستخدام تعليمة التصريح، حيث اسم المتغير يخلق كياناً من نوع البيانات المصرح عنه في تعليمة التصريح ثم يعين له قيمة مناسبة مثال:

```
int a;
a = 12;
```

هاتان التعليمتان تؤديان إلى خلق كيان اسمه a نوعه int وقيمته 12. إذن الكيان a هو وصف لحالة معينة وبإعطائه القيمة 12 ستخزن تلك القيمة في مواقع معينة للتخزين. ولكن في لغة ++C يأخذ مفهوم الكيان بعداً آخر. وهذا البعد هو أي شيء تتعامل معه في البرنامج يمكن اعتباره كياناً. ومن هذا المنطلق فإن للكيان فترة زمنية يمكن له فيها أن يبدأ أو ينشأ ويستمر عمله ثم تنتهي فترته. وللكيان خاصية التعامل مع كيانات أخرى أو يجمع مع كيانات أخرى تحت اسم معين.

إن وضع الكيانات مع بعضها بعضاً بترتيب ما وإعطاءها اسماً معيناً هو عبارة عن تمثيل للصنف (class) . ويُعد الصنف في ++c من أهم ميزات اللغة. وهذه الميزة تجعل لغة ++C لغة برمجة الكيانات (.O.O.P).

11-2-1- تعريف الصنف (class definition):

الصنف هو بنية تحتوي على الأسماء التعريفية والبيانات بأنواعها، وكذلك يحتوي على تعاريف للتوابع. تسمى التوابع المعرفة ضمن بنية الصنف بالتوابع الأعضاء أو العناصر (member function) . كما تسمى البيانات المعرفة ضمن بنية الصنف بالبيانات الأعضاء أو العناصر (member data).

تساعد الأصناف في محاكاة الكائنات التي لها عدة صفات (attributes) (والتي يتم تمثيلها على شكل بيانات أعضاء) . كما يمكنها أن تقوم بتنفيذ عدة أفعال (behaviors) أو عمليات (Operations) (والتي يتم تمثيلها على شكل توابع أعضاء).

وتماماً مثلما تعرف المتغيرات والأسماء، لابدّ من تعريف الصنف قبل استخدامه.

يتم الإعلان عن الصنف بالشكل العام التالي:

```
class class_name
{
    private:
    member variable definition;
    public:
    member function prototype;
};
```

المثال التالي يعرف صنفاً اسمه student:

```
class student
{
    private:
    int student_id;
    double fee;
    public:
    void fees (double);
    int get_id( );
};
```

كما نلاحظ يبدأ تعريف الصنف بالكلمة المفتاحية class ، ثم يلي ذلك اسم الصنف وهو في مثالنا الاسم student وهو اسم مقبول في اللغة. ويعطى من قبل المبرمج ، يتبع ذلك مجموعة من التعاريف المحصورة بين القوسين { } حيث تسمى مجموعة التعاريف هذه بجسم الصنف، وينتهي جسم الصنف بالفاصلة المنقوطة التي تلي آخر قوس في الصنف وهو { .

تعد جميع المتغيرات والتوابع المعرفة ضمن القوسين { } تابعة للصنف. وهذا الأمر يجعل من الصنف وحدة واحدة متكاملة.

يتم في القسم الذي يبدأ بالكلمة المفتاحية private والمتبوعة بالنقطتين ':' التصريح عن كل التوابع الأعضاء والبيانات الأعضاء في الصنف. وتتميز هذه التوابع والبيانات بأنها خاصة، أي لايمكن الوصول إليها من خارج الصنف، بل ينحصر الوصول إليها والتعامل معها من داخل الصنف فقط.

أما في القسم الذي يبدأ بالكلمة المفتاحية public والمتبوعة بالنقطتين ':' فيتم التصريح عن كل التوابع والبيانات الأعضاء العامة في الصنف، وتتميز هذه التوابع والبيانات بأنها متاحة للاستخدام من أية نقطة ضمن البرنامج وتمكن من التعامل مع المعطيات الخاصة للصنف .

تدعى الأسماء public و private بمحددات طريقة الوصول إلى الأعضاء (member access specifiers) . ويمكن أن تظهر عدة مرات في تعريف الصنف ولايوجد أي قيد في ترتيب ظهورها ضمن تعريف الصنف.

ملاحظة (1): لايمكن الإشارة إلى المتغيرات والأسماء والتوابع المعرفة داخل جسم الصنف بشكل مباشر من خارجه كأن نفعل ما يلي على سبيل المثال:

```
class student
{
     private:
     int student_id;
     public:
     void fees(double);
     long get_id ( );
};
student_id=2134;    // error: student_id is undefined
```

أي لايمكن الإشارة إلى متغيرات الصنف خارج حدوده من دون تعيين اسم الصنف والكائن الذي ينتمي إليه.

إن حجم الصنف غير محدد وبالإمكان استعمال أي عدد من المتغيرات والتوابع ضمن الصنف، وهذه المتغيرات قد تشكل أصنافاً أخرى ومؤشرات كيانات من أصناف أخرى أو مؤشرات للذاكرة.

ملاحظة (2): بمقارنة البنية مع الصنف نستنتج أن هناك اختلافاً بين البنية والصنف. أعضاء البنية عامة بشكل افتراضي، بينما أعضاء الصنف خاصة بشكل افتراضي. أي إذا لم نستعمل الكلمات المفتاحية public أو private تكون أعضاء الصنف خاصة. بينما أعضاء البنية عامة، ومن الممكن الوصول إلى جميع عناصرها من دون حماية، وكذلك نلاحظ أن البنية تحتوي على بيانات فقط بينما الصنف يحتوي على بيانات وتوابع تعمل على تلك البيانات وتمكن الوسط الخارجي من الوصول إليها. بكلمات أخرى يمكننا القول إن البنية هي تجمع هامد لبنود البيانات بينما الصنف هو آلية نشطة للبيانات والتوابع. لذا الأصناف أهم بكثير من البنى في c++ . وكما قلنا إن الأصناف تشكل أساس البرمجة الكائنية المنحى. الصنف المعرف سابقاً يحوي على بيانات وهي student_id وهي من نوع int ، وعلى التابعين fees(double) و get_id( ) . وهما غير معرفين بصورة كاملة حيث لانعرف الطريقة التي يؤدي فيها كل تابع عمله، أي ما هي مهمته.

ومن أجل تحديد وكتابة عمل التابع المعرف ضمن الصنف لابدّ من اتباع الخطوات التالية :

1- تسمية الصنف الذي يحتوي على تعريف التابع، وهو هنا الصنف student، وكذلك تحديد نوع المعطيات التي يعيدها أو يحسبها التابع، نلاحظ أن التابع fees لايعيد أية قيمة، لذا تكون القيمة المرجعة من نوع void. ويتبع ذلك العامل (::) الذي يسمى بالعامل الثنائي لتحديد النطاق (binary scope resolution operator) أو العامل الثنائي لتحديد مجال الرؤية. وعلى اعتبار أنه يمكن أن يكون لعدة توابع أعضاء في أصناف مختلفة الاسم نفسه ، تقوم عملية تحديد مجال الرؤية بالربط بين اسم التابع واسم الصنف بحيث نحصل على تحديد وحيد للتوابع الأعضاء المرتبطة بصنف محدد.

حتى لو تم تعريف التابع العضو خارج تعريف الصنف فإنه يبقى خاضعاً لقواعد مجال رؤية الصنف، أي أن اسم ذلك التابع يظل معروفاً من قبل بقية أعضاء الصنف مالم يتم الوصول إليه من خلال كائن من نوع الصنف أو مؤشر على كائن من نوع الصنف نفسه.

إذا تم تعريف تابع عضو ضمن تعريف الصنف، فإنه يأخذ تلقائياً الصفة inline إذا قبل المترجم بذلك . يمكن للتوابع الأعضاء المعرفة خارج تعريف الصنف أن تأخذ الصفة inline بشكل ظاهري بعد استخدام الكلمة المفتاحية inline معها.

يتبع العامل (::) اسم التابع ثم يوضع بين قوسين ( ) أنواع البارامترات الممررة للتابع.

2- كتابة التعليمات التي ستنفذ المهمة المطلوبة من التابع بين قوسين كبيرين { }.

يمكننا الآن كتابة التابع fees بشكل كامل كما يلي:

```
void student :: fees (double fee)
        {
              Fee = 8500.70;
        }
```

10-2-2- مجال رؤية الصنف (class scope):

تقع البيانات الأعضاء (المتغيرات المصرح عنها ضمن تعريف الصنف) والتوابع الأعضاء (التوابع المذكورة ضمن تعريف الصنف أيضاً) ضمن مجال رؤية الصنف. ويتم تعريف التوابع غير الأعضاء ضمن مجال رؤية الملف file scope.

يمكن ضمن مجال رؤية الصنف الوصول مباشرة إلى أعضائه من قبل كافة التوابع الأعضاء التابعة له وذلك بعد ذكرها بالاسم فقط. أما خارج ذلك المجال، فيمكن الوصول إلى أعضاء صنف من خلال اسم كائن (غرض object) أو عنوان كائن أو مؤشر على كائن مرتبط به.

تتمتع التوابع الأعضاء بمجال رؤية (function scope) خاص بها داخل الصنف. يتعرف التابع العضو على المتحولات المصرح عنها ضمن تعريفه فقط. إذا تم تعريف متحول ضمن جسم تعريف تابع عضو باستخدام الاسم نفسه لمتحول مجال رؤية الصنف، فإن مجال رؤية الصنف يقوم بإخفاء المتحول الموجود ضمنه ويعزله عن متحول مجال رؤية التابع باستخدام ما نسميه مجال رؤية التابع. يمكن الوصول إلى المتحول المخفي باستخدام عملية تحديد مجال الرؤية على اسم الصنف. يمكن الوصول إلى المتحولات العامة المخفية باستخدام العملية الأحادية لتحديد مجال الرؤية (::).

11-2-3- الوصول إلى أعضاء الصنف (Accessing class members):

للوصول إلى البيانات والتوابع الأعضاء لصنف نستخدم العمليات المستخدمة نفسها للوصول إلى حقول البنية، أي يمكن الوصول إلى أعضاء صنف باستخدام العملية (.) مصحوبة باسم كائن أو عنوان له. كما يمكن استخدام العملية (<-) مصحوبة باسم مؤشر على كائن للوصول إلى أعضائه.

إذن للوصول إلى البيانات والتوابع الأعضاء لصنف تقوم بعد التصريح عنه بالتصريح عن الكائنات التي تنتمي إليه وتسمى عملية التصريح عن الكائنات

(instantiation) وتشبه هذه العملية من حيث سهولتها التصريح عن المتغيرات. حيث يتم تخصيص مكان له في الذاكرة تشبه ما يتم تخصيصه من خلال التصريح عن المتغيرات من نوع struct. أما التوابع الأعضاء (member function) فلايتم تخصيص مكان لها في الذاكرة ضمن كل غرض، لأن هذه التوابع تستخدم لمعالجة الكائنات الموجودة من نوع الصنف وهي تمثل الواجهة (interface) بين العالم الخارجي وهذه الكائنات الموجودة ضمن الصنف. ويمكن لأي تابع بما في ذلك ( ) main الوصول إلى البيانات والتوابع الأعضاء المعرفة public في الصنف أما الخصائص أي البيانات الأعضاء فلايمكن الوصول إليها إلاّ من قبل التوابع الموجودة معها في الصنف نفسه.

مثال (1) : البرنامج التالي يستخدم صنفاً بسيطاً اسمه display يملك هذا الصنف معطيات عامة x من النمط int وتابعاً عاماً print .ثم يصرح عن ثلاثة متحولات من الصنف display وهي disp و dispRef (عنوان كائن تابع للصنف display) و disptr (مؤشر على كائن تابع للصنف display). المتحول العضو x تم التصريح عنه على أنه عام لتبيان كيفية الوصول إلى هذا النوع من الأعضاء، ويجب أن لاننسى أن البيانات الأعضاء يجب أن تكون عادة بيانات خاصة لتحقيق ميزة إخفاء البيانات.

```
//  Demonstrating the class member access operators. and ->
# include < iostream.h >
//  simple class display
class display
{
    public:
        int x;
        void print ( ) { cout << x << endl; }
};
main ( )
{
    display disp,*disptr;   // create disp object and disptr pointer to disp
    display & dispRef = disp;  // creat reference to disp
    disptr = & disp;
    cout << " Assign 7 to x and print using the objects name:";
    disp . x = 7;     // assign 7 to  data member x,
    disp.print( );     // call member function print.
    cout << " Assign 8 to x and print using a reference:";
    dispRef . x = 8;    // assign 8 to data member x.
    dispRef . print( );   // call member function print.
    cout << " Assign 10 to x and print using a pointer:";
    disptr-> x = 10;     // assign 10 to data member x.
    disptr-> print ( );   // call member function print.
    return 0;
}
```

```
Assign 7 to x and print using the object's name: 7
Assign 8 to x and print using a reference: 8
Assign 10 to x and print using a pointer: 10
```

مثال (2) : البرنامج التالي يصرح عن صنف student له ثلاثة بيانات أعضاء خاصة وتابعين أعضاء عامين. ثم يصرح عن كائنين من صنف student ويعين قيماً لبياناته الأعضاء ثم يستدعي التابع العضو العام print_stdinf الذي يطبع تلك البيانات.

```
// ex102.cpp
# include < iostream.h >
# include < string.h >
// class declaration (specification) start here
class student
{
    private:       // here is the start of private members.
    int std_id;
    char std_name[30];
    float credit;
    public:       // Here is the start of public members.
    student (int, char*,float);
    void print_stdinf(float);
};
// body of the function student ( )
student :: student (int k, char*m, float d)
{
    std_id = k;
    strcpy (std_name,m);
    credit = d;
}
// body of the function print_stdinf
void student :: print_stdinf (float hrs)
{
    cout << " student identification:" << std_id <<"\n"
          << "student name:"<< std_name << "\n"
          << "Fees that must be paid : "<< hrs*credit<<endl;
}
// main program
main ( )
{
    // create some student objects.
    student Ali (23456, "Mohamad Ali",80.0);
    student Samara (23458, "Samara Ali", 20.0);
    Ali.print_stdinf (20.0);
    Samara. print_stdinf (30.0)
    return 0;
}
```

إذن بعد أن نصرح عن صنف ما نستطيع التعامل معه عن طريق التصريح عن كائن من نمط أو نوع الصنف المصرح عنه. في البرنامج السابق تم التصريح عن الصنف student ذي البيانات الخاصة std_id و std_name و credit والتوابع العامة

student( ) و print_stdinf( ) . تم التصريح عن كائن من نوع الصنف student اسمه Ali وعينت لمتغيراته الخاصة القيم الابتدائية التالية:

Std_id		**Std_name**		**credit**
↓	و	↓	و	↓
23456		**Mohamad Ali**		**80.0**

وكذلك التصريح عن كائن آخر Samara وعينت له القيم الابتدائية 23458 و Samara Ali و 20.0 . ثم تم استدعاء التابع print_stdinf مرتين من أجل طباعة المعلومات المدخلة وحساب القسط الواجب دفعه من أجل الساعة المحددة كبارامتر للتابع print_stdinf وسيكون الخرج كالتالي:

```
student identification : 23456
student name : Mohamad Ali
fees that must be paid : 1600
student identification : 23458
student name : samara Ali
Fees that must be paid : 600
```

وهكذا يمكننا استخدام الصنف student لتعريف أي عدد من الكائنات (الأغراض، المتغيرات) من دون إعادة التعريف. ومن المستحسن كتابة تعريف الصنف في ملف رأسي مثلاً (student.h) وجسم الصنف (التعليمات التي تمثل عمل التوابع الأعضاء في الصنف) في ملف آخر مثل (student.cpp) ، وسيكون إعادة استخدامها أفضل علماً أن cpp هو الرمز المستخدم لامتداد الملفات المكتوبة بلغة C++ . سنبين فيما يلي طريقة استخدام الملفات الدليلية بإعادة كتابة البرنامج في المثال (2) في ثلاثة ملفات هي:

1- الملف الرأسي student1.h الذي يمثل مواصفات الصنف student فقط، وهي كالتالي:

```
// student1.h is the name of file
// student class specifications
class student
{
     private:        // private members
     int std-id;
     char std_name[30];
     float credit;
     public :        // public members
        student (int, char*, float);
        void print_stdinf(float);
};
```

2- الملف الذي يحوي تعليمات جسم الصنف هو student1.cpp وهو يحوي التعليمات التي تبين عمل التوابع الأعضاء في الصنف student وهو كالتالي:

```
// student1.cpp is the name of a file which
// contain the body of the class
     # include < iostream.h >
     # include < string.h >
     # include " student1.h "    // class specifications
     student :: student(int k, char*m, float d)
     {
        std_id = k;
        strcpy (std_name,m);
        credit = d;
     }
void student :: print_stdinf (float hrs)
{
     cout << "student identifications : " << std_id << "\n"
           << "student name : " << std_name << "\n"
           << "Fees that must be paid : " << hrs * credit << endl;
}
```

لاحظ أن الملفات الدليلية تم ضمها إلى الملف student1.cpp أعلاه لحاجة ذلك الملف لها وهذه الملفات هي < iostream.h > و < string.h > و " student1.h " .

3- الملف ex103.cpp : الذي يستخدم الصنف student في البرنامج ( )main التالي: لاحظ أننا نحتاج فقط إلى ضم الملف الدليلي "student1.h" إلى البرنامج الرئيسي كما يلي:

```
// ex 103.cpp
// Driver for student class
// Note : compile with student1.cpp
# include " student1.cpp"
main( )
{
    // This is the main program which
    // use the class concepts
    student Ali (23456,"Mohamad Ali",80.0);
    student Samar (23458, "Samar Ali", 20.0);
    Ali . print_stdinf (20.0);
    Samar.print_stdinf (30.0);
    return 0;
}
```

4-2-11- توجيه ما قبل الترجمة # include

لقد تم استخدام التوجيه include # عدة مرات ضمن هذا الكتاب. يفيد هذا التوجيه في تضمين نسخة من ملف معين ضمن الملف المستخدم عند نقطة ذكر التوجيه. تعليمات التوجيه هي عبارة عن تعليمات للمترجم. على عكس تعليمات ++c الاعتيادية، كـ ;int x=3 مثلاً، التي هي تعليمات للمعالج يترجمها المترجم إلى لغة الآلة يستطيع المعالج فهمها. تبدأ جميع تعليمات التوجيه بالرمز (#) دائماً.

يكون للتوجيه include # أحد الشكلين التاليين:

```
# include < filename >
# include " filename "
```

يختلف التوجيهان المذكوران عن بعضهما بعضاً بموقع الملف المطلوب إدراجه. إذا تم إدراج اسم الملف بين إشارتي > و < فهذا يعني أن الملف هو أحد الملفات الرأسية الخاصة بالمكتبة المعيارية للغة. وبالتالي يجب على مرحلة ما قبل الترجمة أن تبحث عنه ضمن مجلدات معروفة مسبقاً ولها علاقة بمكتبات المترجم المعيارية. أما إذا كان اسم الملف مذكوراً بين علامتي تنصيص (") فإن مرحلة ما قبل الترجمة تبحث عنه ضمن المجلد نفسه الذي ينتمي إليه ملف البرنامج المطلوب ترجمته، ثم ضمن المجلدات المعروفة مسبقاً والخاصة بالمترجم. تستخدم الطريقة الثانية (أي تضمين اسم الملف ما بين علامتي تنصيص (" ") لإدراج الملفات الرأسية المكتوبة من قبل المستثمر.

لقد استخدمنا التوجيه include # حتى الآن عدة مرات كما ذكرنا وذلك لإدراج ملفات رأسية معيارية مثل math.h, string.h, iomanip.h, iostream.h .

مثال (3) : البرنامج التالي يصرح عن صنف اسمه time ثم ينشأ كائنين من صنف time هما t1 و t2 ويطلب إدخال بيانات لهذين الكائنين ثم طباعتها.

```
# include < iostream.h >
class time
{
    private :
    int hours;
    int minutes;
    public:
    void enter( );
    void display( );
};
main( )
{
    time t1,t2;
    cout << " enter member variable for t1:";
    t1.enter ( );
    cout << " enter member variable for t2:";
    t2.enter ( );
    cout << "\n display t1 as hour:minute"<< endl;
    cout << " t1 = ";
    t1.display( );
    cout << "\n display t2 as hour:minute"<< endl;
    cout << " t2 = ";
    t2.display( );
    return 0;
}
void time :: enter( )
{
    char x;
    cout << "enter time (format h:m):";
    cin >> hours >> x >> minutes;
}
void time :: display ( )
{
    cout << hours << ':' << minutes << endl;
}
```

خرج البرنامج هو :

```
enter member variable for t1:enter time(format h:m):10:20
enter member variable for t2 :enter time(format h:m):8:39
display t1 as hour : minute
t1 = 10:20
display t2 as hour : minute
t2 = 8:39
```

لقد تم الإعلان عن الصنف time الذي يملك بيانات أعضاء خاصة (يتم الوصول إليها من داخل الصنف فقط) هما المتحولان hour و minute ، وكذلك توابع أعضاء عامة (يتم الوصول إليها من خارج الصنف) وهي enter( ) و display( ). ثم الإعلان في البرنامج الرئيسي main عن كائنين t1 و t2 من نوع الصنف time

وبالتالي هذه الكائنات لها بيانات خاصة هي minute,hour وتوابع عامة تستطيع الوصول لها هي ( )enter و ( )display . طريقة الوصول لهذه التوابع هي كما ذكرنا باستخدام النقطة (٠) مابين اسم الكائن واسم التابع كما يلي.

```
t1.enter( );
t1.display( );
```

لقد أعلن عن التوابع داخل الصنف في حين جرى تعريفها خارجه. وعند تعريف التوابع خارج الصنف لابدّ من كتابة نوع القيمة المعادة أولاً ثم اسم الصنف ومن ثم معامل تحديد المجال (::) ومن ثم اسم التابع ومعاملاته والتعليمات التي تشكل جسمه ما بين قوسين كبيرين { } . من المهم جداً استخدام معامل تحديد المجال (::) حتى يتمكن المترجم من معرفة أن التابع الجاري تعريفه هو تابع عضو في الصنف المذكور اسمه قبل المعامل (::).

11-2-5- تعليمات التعيين مع الكائنات:

إن من مزايا استعمال الأصناف هي مقدرتها على تعيين قيمة أحد الكائنات إلى كائن آخر من الصنف نفسه. نحن نعلم كيف يتم التعيين مع الأنواع الأساسية للبيانات كـ int مثلاً :

```
int var = 3;
```

يعطى القيمة 3 للمتغير var . بالإمكان فعل الشيء نفسه مع الكائنات من أنواع أصناف عرفت من قبل المستثمر. لنقم بتعديل البرنامج في المثال (3) بحيث نحصل في التابع main على قيمة الكائن t1 ذي الصنف time ثم نعين هذه القيمة إلى الكائن t2 بوساطة التعليمة التالية :

```
t2 = t1;
```

ثم نقوم بعرض بيانات الكائن t2 باستخدام العلاقة ()t2.display . عمل كل من التابع ()enter والتابع ( )display هو كما في المثال (3) ، ويصبح البرنامج كالتالي:

مثال (4):

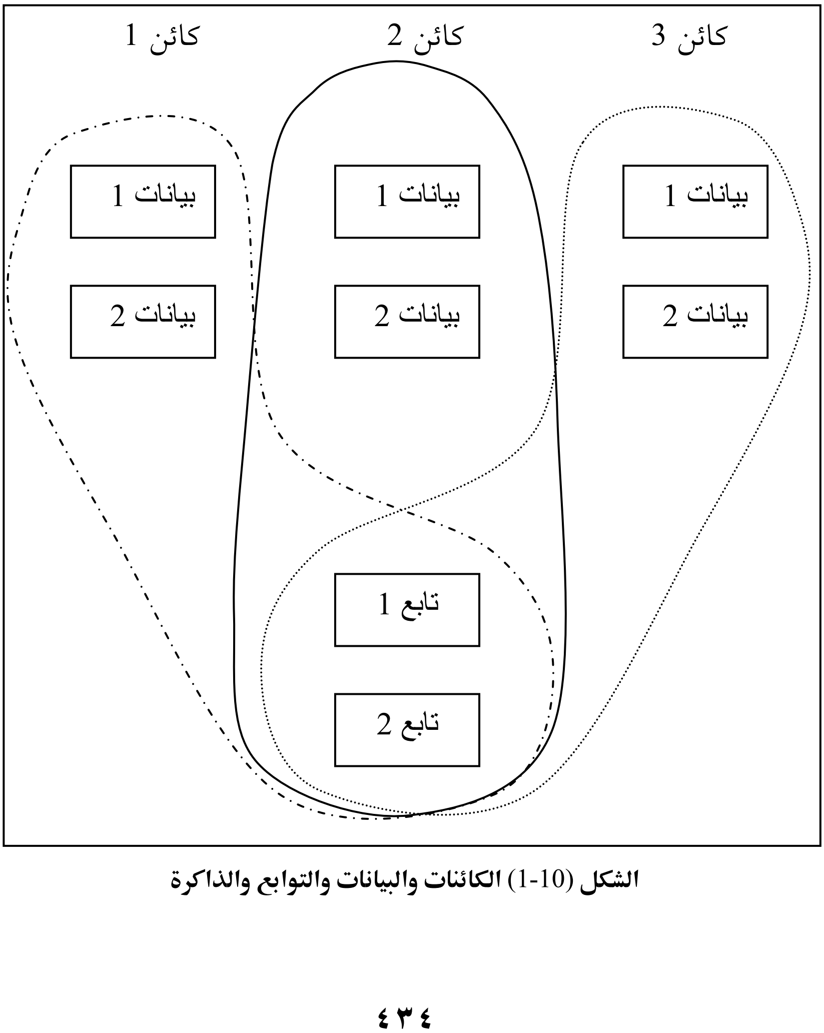
```
# include < iostream.h >
class time
{
    private:
    int hours;
    int minutes;
    public:
    void enter( );
    void display( );
};
main( )
{
    time t1, t2;     // create two time objects (variables)
    cout << " enter member variable for t1:";
    t1. enter( );
    t2 = t1;         // make t2 equal to t1
    cout << "display t2 as hour : minute "<< endl;
    cout << "t2 = ";
    t2 . display( );    // display t2
    return 0;
}
```

تمتاز C++ بالقدرة على جعل أحد الكائنات مساوياً للآخر بالرغم من وجود أكثر من بند معطيات في الكائن. في مثالنا السابق، الكائن t1 يحوي بندي بيانات (hours, minutes) ، ولن يعترض المترجم بشأن نقل هاتين القيمتين من كائن إلى آخر. مهما كان عدد البيانات الموجودة في الكائن الأول سيتم نسخها إلى الكائن الثاني في أثناء تنفيذ تعليمة التعيين. ولكن السؤال ماذا يحدث للتوابع الأعضاء؟ هل يتم نسخها هي أيضاً.من السهل الاعتقاد بذلك. أي كل شيء مرتبط بالكائن الأول يتم نسخه إلى الكائن الثاني، ولكن في الحقيقة هناك نسخة واحدة من كل تابع عضو في الذاكرة ،

مهما كان عدد الكائنات المنشأة من ذلك الصنف. وهذا منطقي كون التوابع الأعضاء هي نفسها مع كل الكائنات. على عكس البيانات التي تحتوي عادة على قيم مختلفة لكل كائن. وسيكون تبذيراً للذاكرة لو تم نسخ التوابع الأعضاء. لذا تتشارك كل الكائنات التوابع الأعضاء كما هو مبين في الشكل (10-1) . عندما تستدعي تابع عضو لكائن ما، كما في التعليمة التالية :

```
t2 . display( );    // display t2.
```

سيتم دائماً استدعاء التابع ( )display . لكنها تعمل على بيانات مختلفة وفقاً للكائن الذي استدعاها. في مثالنا السابق سيعمل التابع ( )display في هذه الحالة على البيانات المخزنة في t2.



الشكل (10-1) الكائنات والبيانات والتوابع والذاكرة

6-2-11-توابع البناء وتوابع الهدم في الصنف

(Constructors and destructors functions in classes)

عند إنشاء كائن ما يمكن لبياناته الأعضاء أن تأخذ قيماً ابتدائية بوساطة التابع الباني المرتبط بالصنف، يحمل التابع الباني اسم الصنف نفسه ويتم استدعاؤه بشكل تلقائي عند إنشاء أو اشتقاق أي كائن من الصنف (instantiated) . لايمكن إعطاء قيم ابتدائية للبيانات الأعضاء ضمن تعريف الصنف، وإنما يجب القيام بذلك باستخدام التابع الباني المرتبط بالصنف، ويمكن تعديل هذه القيم لاحقاً بعد إنشاء الكائن. كما لايمكن أن تحدد التوابع البانية أية أنواع أو قيم معادة ولاحتى void ، ويمكن تطبيق عمليات تحميل زائد عليها من أجل إعطاء قيم ابتدائية متعددة لكائنات الصنف. إذا لم يتم التصريح بشكل واضح عن التابع الباني فإن المترجم يولد الشيفرة الضرورية لإنشاء الكائن وتخصيص الذاكرة له ويجري أنواعاً أخرى من أعمال التخصيص ويتم هذا الأمر تلقائياً (default constructor) مثلما يتم للمتغيرات ذات الأنواع الأساسية كـ int مثلاً.

عند التصريح عن كائن مرتبط بصنف ما ، فإنه يمكن تحديد القيم الابتدائية الممكن إعطاؤها له ضمن قوسين على يمين اسم الكائن وقبل الفاصلة المنقوطة. تمرر هذه القيم إلى التابع الباني لتغدو كبارامترات له.

مرّ بنا في المثال (2) تابع بانٍ يحمل اسم الصنف student نفسه وله ثلاثة بارامترات d , *m , k . وقد تم إعطاء قيم ابتدائية لكل من هذه البارامترات عند الإعلان عن الكائن Ali من الصنف student وذلك بالتعليمة التالية:

```
student Ali (23456 , " Mohamad Ali ", 80.0);
```

وبذلك تأخذ البيانات الأعضاء الخاصة القيم الابتدائية التالية:

```
std_id = 23456
std_name = Mohamad Ali
credit = 80.0
```

إذن يستخدم التابع الباني لتخصيص البيانات الأعضاء للكائنات قيماً ابتدائية، وسنأخذ مثالاً لبانٍ بأكثر من بارامتر وبانٍ آخر ذي بارامتر واحد وباني من دون بارامترات. لكن قبل ذلك لنشرح ما هو التابع الهادم ؟

يحمل التابع الهادم اسم الصنف المعرف نفسه ضمنه مسبوقاً بإشارة ~ . ولايرجع هذا التابع قيمة ولايأخذ أي بارامتر. يقوم هذا التابع بإزالة أو إلغاء تخصيص مساحة الذاكرة التي كان التابع الباني قد خصصها للكائن. وسواء كتب المبرمج تابعاً هادماً أم لا، يزود المترجم بروتين خاص لإلغاء تخصيص الذاكرة التي كان يحتلها الكائن (default destructor).

مثال (5) : البرنامج التالي يبين كيف يعمل التابع الباني والهادم:

```
# include < iostream.h >
class Omega
{
    public:
       Omega( )
       {
          cout << " I am the constructor."<<endl;
       }
       ~Omega( )
          {
              cout<<"I am the destructor."<<endl;
          }
};
main( )
{
    cout<<"starting main( )."<<endl;
    Omega om1, om2, om3;
    cout << " ending main( )."<<endl;
    return 0;
}
```

خرج البرنامج:

```
starting main( ).
I am the constructor.
I am the constructor.
I am the constructor.
ending main( ).
I am the destructor.
I am the destructor.
I am the destructor.
```

نلاحظ أن التابع الباني لايقوم بأي تخصيص لكنه يعرض رسالة تعلن أنه يتم تنفيذه، كذلك الأمر بالنسبة للتابع الهادم. كما نلاحظ أنه لايوجد استدعاءات للتوابع ضمن التابع main( ) حيث إن الباني والهادم يتم استدعاؤهما تلقائياً. فالباني يتم استدعاؤه تلقائياً كلما تم إنشاء كائن جديد.

ولقد تم في البرنامج السابق استدعاء التابع الباني ثلاث مرات لأنه تم إنشاء ثلاثة كائنات om1 , om2 , om3 . عند انتهاء التابع main( ) ينتهي مدى الكائنات الثلاثة ويتم تدميرها تلقائياً حيث يتم استدعاء التابع الهادم لكل كائن قبل تدميره كما يظهر ذلك في الخرج إذ إن عبارات التابع الهادم تظهر بعد انتهاء التابع main( ).

مثال (6) : نكتب الصنف Rational باستخدام تابع بانٍ وآخر هادم.

```
# include < iostream.h >
class Rational
{
    public:
    Rational( ) { cout<< "Object is born.\n"; }
    ~ Rational( ) { cout<< "Object dies.\n"; }
    private:
    int num, den;
};
main( )
```

```
{
    {
       Rational x;     // beginning of scope for object x
       cout << " Now x is alive. \n";
    }  // end of scope for object x.
    cout << " Now between blocks. \n";
    {
       Rational y;     // beginning of scope for object y
       cout << " Now y is alive. \n";
    }  // end of scope for object y
    return 0;
}
```

الخرج هو :

```
Object is born.
Now x is alive.
Object dies.
Now between blocks.
Object is born.
Now y is alive.
Object dies.
```

يستدعى هادم الصنف عند وصول الكائن إلى نهاية مجاله. بالنسبة لكائن محلي سيكون هذا عند نهاية المجموعة (block) التي أعلن خلالها. وبالنسبة للكائن static ستكون عند نهاية main .

مثال (7) : البرنامج التالي يعلن صنف Rational له بيانات أعضاء خاصة num و den وتوابع أعضاء عامة assign( ) و convert( ) و invert( ) و print( ) ثم يصرح عن كائن x من الصنف Rational ويستدعي التوابع الأعضاء لتنفيذ تعليماتها المطلوبة:

```
# include < iostream.h >
class Rational
{
     public:
     void assign (int,int);
     double convert( );
     void invert( );
     void print( );
     private:
     int num, den;
};
void Rational :: assign (int n, int d)
{
     num = n;
     den = d;
}
double Rational :: convert( )
{
     return double (num) / den;
}
void Rational :: invert( )
{
     int temp = num;
     num = den;
     den = temp;
}
void Rational :: print( )
{
     cout << num << '/' << den;
}
main()
     {
        Rational x;
        x.assign (22,7);
        cout << " x = ";
        x . print( );
```

```
    cout << " = " << x . convert( ) << endl;
    x. invert( );
    cout << "1/x = ";
    x . print ( );
    cout << endl;
 return 0;
  }
```

والخرج هو :

```
x = 22/7 = 3.14286
1/x = 7/22
```

تم الإعلان في البرنامج عن الكائن x من الصنف Rational وبالتالي لها بيانات أعضاء خاصة بها هي num و den ولها القدرة على استدعاء التوابع الأعضاء العامة الأربعة ( )print( ), convert( ), invert( ), assign .

التابع ( )assign	يعطي قيماً ابتدائية لبيانات الصنف.
التابع ( )convert	يُوجِد ناتج القسمة الحقيقي لـ num/den
التابع ( )invert	يقوم بالتبديل بين num و den
التابع ( )print	يطبع الكسر المكون من البيانات الأعضاء num و den

لقد تم استدعاء التابع بذكر اسمه وإلحاقه بالكائن بوساطة الرمز (.) . كما تم ربط تعريف التابع بذكر اسم التابع يسبقه عامل تحديد النطاق :: واسم الصنف الذي يتبع له هذا التابع.

لقد تم استعمال التابع ( )assign لإعطاء قيم ابتدائية للبيانات. ولقد رأينا أن لغة ++c تسمح بإعطاء قيم أولية لبيانات الكائن عند الإعلان عنه باستخدام التابع الباني والذي يحمل اسم الصنف نفسه ويستدعى تلقائياً عند الإعلان عن الكائن. سنعيد كتابة المثال (7) باستعمال التابع الباني. وكتابة تعريف التوابع ضمن تعريف الصنف.

مثال (8): البرنامج التالي يعيد كتابة البرنامج في المثال (7) ولكن باستخدام التابع الباني :

```
# include < iostream.h >
class Rational
{
     public:
     Rational (int n, int d) : num (n), den  (d) { }
     double convert ( ) { return double (num)/den; }
     void invert( ) { int temp = num; num = den; den = temp; }
     void print( ) { cout << num << ' / ' << den; }
     private:
     int num, den;
};
main( )
{
         Rational x (22,7);
         cout << " x = ";
         x . print ( );
         cout << "="  << x . convert ( )<< endl;
         x . invert ( );
         cout << "1/x=";
         x . print ( );
         cout << endl;
         return 0;
}
```

الخرج هو :

```
x = 22/7 = 3.14286
1/x 7/22
```

التابع الباني ( )Rational له تأثير التابع ( )assign نفسه. إنه يبدأ الكائن عن طريق إعطائه القيم المحددة للبيانات الأعضاء. عند تنفيذ الإعلان عن الكائن x ،

يستدعى التابع الباني تلقائياً والأرقام الصحيحة 22 و 7 تمرر لبارامتر n و d . وعندئذ يعطي التابع هذه القيم إلى البيانات الأعضاء الخاصة بالكائن x حيث num = n و den = d.

إن الإعلان :

```
Rational x (22,7);
```

يكافئ التعليمتين التاليتين :

```
Rational x;
x . assign (22,7);
```

يمكن كما ذكرنا التحميل الزائد لاسم التابع الباني في الصنف. ويتم التفريق بينهما عن طريق قوائم البارامترات المحددة.

11-2-6-1- التحميل الزائد للتابع الباني والمتحولات الافتراضية:

يمكن في لغة C++ أن يحتوي البرنامج عدة توابع تحمل الاسم نفسه كما ذكرنا سابقاً في الفصل الثامن (التوابع). وهذا يسمح للمبرمج باستخدام اسم تابع واحد في ظروف وحالات مختلفة، كما يمكن إعطاء قيم افتراضية لباراميترات التابع وهذا يمكن من تعريف عدة توابع بناء (Constructor) في الصنف نفسه. أي نستطيع أن نستخدم توابع بناء مختلفة في حالات مختلفة ويتعرف المترجم (compiler) على كل تابع من خلال عدد الباراميترات المستخدمة للتصريح عنه. فمثلاً عند تعريف صنف ما وليكن Inventory ، قد يحتاج المبرمج إلى إنشاء كيان (فقرة مخزون) يحوي قيم جميع الخصائص الموجودة في ذلك الكيان، وقد يحتاج في ظروف أخرى إلى إنشاء كيان يحوي فقط رقم المادة وإعطاء قيم افتراضية لبقية الخصائص. كما قد يحتاج في حالة ثالثة إلى إنشاء حساب يحتوي فقط رقم المادة والكمية المتوفرة ويعطي قيمة افتراضية لباقي الخصائص.

في مثل هذه الحالات يجب التصريح عن ثلاثة توابع بناء على النحو التالي:

```
class Inventory
{
    private:
      int itemNo;
      double qtyOnHand;
      double qtyReserved;
    public:
      Inventory (int, double, double);
      Inventory (int, double);
      Inventory (int);
};
```

أما تعريف توابع البناء الثلاث السابقة فيكون على النحو التالي:

```
Inventory : : Inventory (int id, double aqty, double rqty)
     {
       itemNo=id;
       qtyOnHand=aqty;
       qtyReserved=rqty;
     }
Inventory : : Inventory( int id, double aqty)
     {
       itemNo=id;
       qtyOnHand=aqty;
       qtyReserved=80.8;
     }
Inventory : : Inventory (int id)
     {
       itemNo=id;
       qtyOnHand=127.6;
       qtyReserved=26.7;
     }
```

نلاحظ مما سبق أن القيمة الافتراضية للباراميترات يجب تحديدها ضمن تعريف التابع. وأن جميع نسخ التوابع يجب أن ترجع قيمة من النوع نفسه (return type)، ويجب أن تختلف في عدد أو أنواع الباراميترات التي تأخذها عند استدعائها.

مثال (9): لنعيد كتابة المثال (7) باستخدام أكثر من تابع باني للصنف Rational:

```
# include < iostream.h >
class Rational
{
    public:
    Rational( ): num(0), den(1) { }
    Rational(int n): num(n), den(1) { }
    Rational (int n, int d): num(n), den(d) { }
    void print( ) { cout << num << '/' << den; }
    private:
    int num, den;
};
main( )
{
    Rational x, y(4), z(22,7);
    cout << " x = ";
    x . print( );
    cout << "\n y = ";
    y. print( );
    cout << " \n z = ";
    z. print();
    return 0;
}
```

والخرج هو :

```
x = 0/1
y = 4/1
z = 22/7
```

في المثال السابق (9) ثلاثة توابع بانية. الأول ليس له بارامترات ويبدأ الكائن x بالقيم الافتراضية 0 و 1. الباني الثاني له بارامتر واحد وهو العدد الصحيح 4 ويبدأ

الكائن y ليكون الكسر المكافئ لهذا العدد وهو 4/1 . الباني الثالث له بارامتران ويبدأ الكائن z بالقيم 22 و 7 كما في المثال (7).

الباني الأبسط من بين البانيات التي قد يتضمنها الصنف هو الباني الذي لايحوي بارامترات ويسمى الباني الافتراضي. وإذا لم يعلن هذا الباني الافتراضي في توصيف الصنف فإن النظام سيوجده تلقائياً للصنف.

لاحظ التعريفين التاليين للتابع الباني :

```
Rational (int n, int d) { num = n; den = d; }
Rational (int n, int d): num(n), den(d) { }
```

أوامر التعريف في جسم التابع التي عرفت n على أنه num و d على أنه den تم حذفها في الشكل الثاني. وتم الحصول على التأثير نفسه عن طريق ما يسمى بقائمة البدء الموضحة والتي تأتي بعد الرمز (:) وتتقدم جسم التابع والذي هو فارغ { }.

11-2-6-2- استخدام القيم الافتراضية للبارامترات في التابع الباني:

يمكن الاستغناء عن التوابع البانية الثلاثة السابقة، وضمها في تابع باني واحد باستخدام القيم الافتراضية للبارامترات كما يلي:

مثال (10): البرنامج التالي يستخدم تابعاً بانياً مع قيم افتراضية للبارامترات :

```
# include < iostream.h >
class Rational
{
    public:
       Rational (int n = 0, int d=1): num(n), den(d) { }
       void print( ) { cout << num<< '/' << den; }
    private:
    int num, den;
};
main( )
```

```
{
    Rational x , y(4) , z(22,7);
    cout << " x = ";
    x . print( );
    cout << " \n y = ";
    y . print( );
    cout << " \n z = ";
    z . print( );
    return 0;
}
```

والخرج هو :

```
x = 0/1
y = 4/1
z = 22/7
```

القيم الافترا ضية تستخدم عندما لاتمرر بارامترات حقيقية. لذا في الإعلان عن الكائن x لم نمرر له بارامترات، البارامتر الأساسي n تم إعطاؤه القيمة الافتراضية 0 والتي أعطيت بعدها إلى x . num والبارامتر الأساسي d أعطي القيمة الافتراضية 1 والتي نسبت بعدها إلى x.den.

أما في الإعلان عن الكائن y فقد مرر له بارامتر واحد هو 4 وبذلك أخذ البارامتر n القيمة 4 والتي نسبت بعد ذلك إلى y.num ، أما البارامتر d فقد أعطي القيمة الافتراضية 1 والتي نسبت فيما بعد إلى y.den . في الإعلان عن z لم تستخدم القيم الافتراضية لأنه تم الإعلان عنه باستخدام قيمتين للبارامترات الخاصة به.

التابع الباني له بارامتران هما n و d يأخذان قيمتين تلقائيتين هما 0 لـ n و 1 لـ d وفي لائحة التخصيص تم تخصيص n لـ num و d لـ den.

11-2-6-3- التابع الباني الناسخ (The copy constructor):

تتم عملية نسخ قيمة كائن إلى كائن آخر من الصنف نفسه باستخدام التعليمات التالية :

```
time t1, t2;
t1. getdata( );
t2 = t1;
```

علامة = لاتعني المساواة هنا وإنما تخصص أو تنسب البيانات التي وضعها التابع getdata( ) في t1 إلى الكائن t2 عند تنفيذ التعليمة الأخيرة.

يمكن كتابة ما سبق كما يلي:

```
time t1;
t1 . getdata( );
time t2 (t1);
```

يسمى هذا النوع من التخصيص بالتدوين التابعي.

مثال (11): البرنامج التالي يصرح عن صنف time له بيانات أعضاء hour و minute وهي خاصة . وتابع باني ( )time له بارامتران وتابع للإظهار ( )display وهما تابعان عامان. كذلك يصرح عن متغيرين صحيحين ومتغير آخر محرفي ويطلب إدخال قيم لهما في التابع ()main ثم يصرح عن كائن t من صنف time ويعين له القيم المدخلة ثم يعرضها وبعد ذلك ينسخها إلى كائن آخر t1 ويعرض بياناته. كما ينسخ بيانات الكائن t1 إلى كائن t2 من الصنف time ويظهر بياناته.

```
# include < iostream.h >
# include < iomanip.h >
class time
{
    private:
    int hour;
    int minute;
    public:
    time (int h, int m): hour(h), minute(m) { }
    void display( )
    {
       cout << hour << ':' << setfill ('0') << setw(2)
              << minute << endl;
    }
};
main()
{
     int hh, mm;
     char ch;
     cout << " for t, enter time (format 23:59):";
     cin >> hh >> ch >> mm;
     time t(hh, mm);
     cout << "t=";
     t . display( );
     time t1 = t;
     cout << "t1=";
     t1.display( );
     time t2(t1);
     cout << "t2=";
     t2.display( );
     return 0;
}
```

والخرج هو :

```
for t, enter time (format 23:59): 10:45↵
t = 10 : 45
t1 = 10 : 45
t2 = 10 : 45
```

لقد استخدم البرنامج طريقتين في النسخ، الأولى باستخدام التخصيص:

`time t1 = t;`

والثانية باستخدام طريقة التدوين التابعي:

`time t2 (t1);`

في المثال التالي سنقوم بنسخ كائن إلى كائن آخر من الصنف نفسه باستخدام التابع الباني الناسخ.

يقوم التابع الباني الناسخ الافتراضي الذي ينشئه المترجم إذا لم يكتب المبرمج بانياً ناسخاً، يقوم بالنسخ العضوي لكل البيانات من كائن لآخر. والمقصود بالنسخ العضوي أن الباني الناسخ الافتراضي لايقوم بأي جهد لتغيير أو تفسير البيانات. كل ما يفعله هو نسخ قيمة كل عضو إلى العضو المرادف في الكائن الآخر.

التابع الباني التلقائي (الافتراضي) ()time للصنف time يستدعى تلقائياً عند إعلان كائن من نوع الصنف time في أبسط صورة مثل:

```
time t1(t)
```

أما الباني الناسخ فيتم الإعلان عنه بالشكل:

```
time (const time &);
```

للباني الناسخ بارامتر واحد وهو الكائن الذي تنسخه، هذا الكائن يمرر بالمرجع الثابت حيث إنه يجب أن لايتغير. عند استدعاء الباني الناسخ فإنه ينسخ الحالة الكاملة للكائن في كائن جديد من الصنف نفسه. وكما ذكرنا إذا كان تعريف الصنف لايتضمن صراحة بانياً ناسخاً عندها يخلق النظام واحداً تلقائياً، وهذا كافٍ في أغلب الأحيان، ولكن لو أردنا أن يكون الكائن الجديد مختلفاً عن الكائن الذي ننسخ منه بطريقة ما، أو عندما نريد أن يكون لعملية النسخ تأثيرات جانبية ما. عندها يجب تعريف بانياً ناسخاً خاصاً يقوم بذلك.

للتابع الباني الناسخ اسم الصنف نفسه وليس له قيمة مرجعة . وبارامتره دائماً من نوع الصنف نفسه الذي يكون التابع الباني عضواً فيه ويمثل البارامتر الكائن المطلوب نسخه.

إن تمرير البارامتر للباني الناسخ يتم بالمرجع ويتم الإعلان عنه كثابت لنضمن بذلك عدم إمكانية تعديل البارامتر. أما لو مرر البارامتر بالقيمة فإننا نضمن أن

البارامتر الأصلي يبقى محمياً من أي تغيير، إلاّ أن هذا يسبب نسخاً جديدة من الكائن ويسبب استهلاكاً للذاكرة، وتحتاج عملية النسخ هذه للكثير من الوقت وقد تسبب بعض المشاكل أيضاً.

مثال (12): البرنامج التالي يوضح أن أية محاولة لتغيير بارامتر صرح عنه أنه ثابت يعطي خطأ وبذلك تتم حماية البارامترات من التغيير :

```
// demonstrates passing const variable
# include < iostream.h >
class alpha
{
     private:
     int alphadata;
     public:
     void getdata (const int & i)     // guarantees not to change i
     {
        alphadata =i;            // this is ok
        i = 44;                  // error: cannot modify const variable
     }
};
main( )
{
     alpha  alph;
     int ivar = 80;    // I don’t want getdata( ) to modify ivar
     alph . getdata (ivar);    // perfectly safe
     cout << " alph data = " << alph . getdata (ivar) << endl;
     return 0;
}
```

هنا أريد أن أضمن أن التابع العضو getdata( )، عند استدعائه من التابع main( ) مع ivar كبارامتر بمرجع، لن يعدل ivar. لتحقيق ذلك . تم جعل البارامتر في getdata( ) ثابتاً.

مثال (13): البرنامج التالي يستخدم تابعاً بانياً ناسخاً صراحة بدل استخدام الباني الناسخ التلقائي كما في المثال (11).

```
# include < iostream.h >
# include < iomanip.h >
class time
{
    private:
    int hour;
    int minute;
    public:
    time (int h, int m): hour(h), minute(m) { }
    time (const time & ti): hour (ti.hour), minute (ti. minute) { }
    void display( )
    {
       cout << hour << ':' << setfill ('0') << setw(2) << minute << endl;
    }
};
main()
{
    int hh, mm;
    char ch;
    cout << " for t, enter time (format 23:59):";
    cin >> hh >> ch >> mm;
    time t(hh, mm);
    cout << "t=";
    t . display( );
    time t1 = t;
    cout << "t1=";
    t1.display( );
    time t2(t1);
    cout << "t2=";
    t2.display( );
    return 0;
}
```

والخرج هو :

```
for t, enter time (format 23:59): 10:45
t = 10 : 45
t1 = 10 : 45
t2 = 10 : 45
```

مثال (14): البرنامج التالي يضيف للتابع الباني الناسخ تأثيرات جانبية مرغوبة بوجود عداد في المتغير الساكن number.

```
# include < iostream.h >
# include < iomanip.h >
class time
{
    private:
    int hour;
    int minute;
    static int number;
    public:
    time (int h, int m): hour(h), minute(m) { }
    time (const time & ti): hour (ti.hour), minute (ti. minute)
    {
       cout << " i am the copy constructor " << ++ number << endl;
    }
    void display()
    {
       cout<< hour <<':' << setfill ('0') << setw(2)
             << minute << endl;
    }
 };
 int time :: number = 0;
 main ( )
  {
    int hh, mm;
    char ch;
    cout << " for t, enter time (format 23:59):";
    cin >> hh >> ch >> mm;
    time t(hh, mm);
    cout << "t=";
    t. display( );
    time t1 = t;
    cout << "t1=";
    t1.display( );
    time t2(t1);
    cout << "t2=";
    t2.display( );
    return 0;
}
```

والخرج هو :

```
for t, enter time (format 23:59): 10:30↲
t = 10 : 30
i am the copy constructor 1
t1 = 10 : 30
i am the copy constructor 2
t2 = 10 : 30
```

تم الإعلان عن المتغير عضو البيانات الخاص number على أنه static أي أنه يحافظ على آخر قيمة وصل إليها. وتم إعطاؤه قيمة ابتدائية خارج الصنف time من خلال التعليمة :

`int time :: number = 0;`

حيث حدد العامل :: أن عضو البيانات number هو ضمن مجال الصنف time ولقد تم زيادة المتغير number كل مرة يتم فيها استدعاء التابع الباني الناسخ لمعرفة عدد الكائنات المنسوخة عن الكائن t .

7-2-10- توابع الوصول (Access functions) :

رغم أن البيانات الأعضاء في الصنف يتم الإعلان عنها على أنها خاصة للحد من الوصول إليها. إلاّ أنه يتم وضع توابع أعضاء عامة تسهل فقط قراءة البيانات الأعضاء الخاصة . تسمى هذه التوابع بتوابع الوصول.

مثال (15)

```
# include < iostream.h >
class Rational
{
    public:
    Rational (int n = 0, int d = 1): num(n), den(d) { }
    int numerator ( ) const { return num; }
    int denominator ( ) const { return den; }
    private:
    int num, den;
};
main ( )
{
    Rational x (22,7);
    cout << x. numerator ( ) << '/' << x. denominator ( )
          << endl;
}
```

والخرج هو :

```
22/7
```

التوابع الأعضاء ( ) denominator ( ) , numerator تعيد القيم لبيانات الكائن x الخاصة.

8-2-11- التوابع الأعضاء الخاصة :

يتم عادة الإعلان عن البيانات الأعضاء على أنها خاصة، والتوابع الأعضاء على أنها عامة. ولكن في بعض الحالات يكون من المفيد الإعلان عن تابع عضو أو أكثر على أنه خاص. وبذلك يمكن استخدام مثل هذا التابع داخل الصنف نفسه. أي أنها توابع خدمة محلية.

مثال (16): البرنامج التالي يصرح عن صنف Rational له تابع بانٍ وآخر للطباعة وهما توابع أعضاء عامة، وله بيانات أعضاء خاصة وتابعين أعضاء خاصين كذلك. الأول gcd( ) يعيد قيمة القاسم المشترك الأعظم لعددين صحيحين مررا له، والتابع reduce ( ) يستخدم التابع gcd لاختزال الكسر إلى أقل درجة. بحيث يختصر الكسر 100/360 إلى 5/18.

```
# include < iostream.h >
  class Rational
{
    public:
    Rational (int n = 0, int d = 1): num(n), den(d) { reduce( ); }
    void print ( ) { cout << num << '/' << den << endl;}
    private:
    int num, den;
    int gcd (int j, int k)
    {
       if (k = = 0)
       return j;
       return gcd (k, j% k);
    }
void reduce ( )
    {
       int g = gcd (num, den);
       num/= g;
       den /= g;
    }
};
main ( )
{
    Rational x (100, 360);
    x. print( );
}
```

والخرج هو :

```
5/18
```

نلاحظ في البرنامج استدعاء التابع ( ) reduce من قبل التابع الباني Rational لأنه كما ذكرنا سابقاً لايمكن استدعاء التوابع الأعضاء الخاصة من التابع ( )main وإنما يجب استدعاؤها من داخل الصنف الذي عرف فيه.

11-2-9- التوابع الأعضاء العامة في الصنف:

(The public member function in class)

يمكن أن تأخذ التوابع الأعضاء في الصنف عدة أشكال كما يلي:

1- توابع أعضاء من دون باراميترات.

2- توابع أعضاء مع باراميترات.

3- توابع أعضاء تغير من قيم البيانات الأعضاء.

4- توابع أعضاء تجري عمليات حسابية على البيانات الأعضاء.

5- توابع أعضاء ترجع قيماً من أي نوع معروف في اللغة. كما يمكن أن تكون القيمة المرجعة من نوع صنف.

6- توابع أعضاء تستدعي بعضها. كما يمكن استدعاء التوابع الأعضاء بالمرجع.

11-2-9-1- التوابع الأعضاء العامة في الصنف من دون بارامترات:

مثال (17) : البرنامج التالي يوضح تعريف الصنف time كما يعرف التوابع الأعضاء في الصنف ضمن تعريف الصنف ثم ينشأ كائنين ويعين لهما قيماً ثم يطبعهما.

```
# include < iostream.h >
class time
{
    private:
    int hour, minute;
    public:
    void enter( )
    {
       char ch;
       cout << " enter time (format 23:59):";
       cin >> hour >> ch >> minute;
    }
    void display( )
    { cout << hour << ":" << minute << endl; }
};
main ( )
{
    time t1, t2;
    cout << " for t1,";
    t1 . enter ( );
    cout << " for t2,";
    t2 . enter( );
    cout << "\n t1 = ";
    t1 . display ( );
    cout << "\n t2 = ";
    t2 . display( );
    return 0;
}
```

خرج البرنامج :

```
for t1, enter time (format 23:59): 10:30
for t2, enter time (format 23:59): 14:15
t1 = 10:30
t2 = 14:15
```

11-2-9-2- التوابع الأعضاء العامة في الصنف مع باراميترات:

في بعض الأحيان من المفيد ترك مسؤولية إعطاء قيم للبيانات الأعضاء للمستخدم عن طريق التابع main( ) ومن ثم تمريرها إلى التابع العضو كبارامترات.

مثال (18) : البرنامج التالي يصرح عن صنف time ويطلب من المستخدم إدخال قيم ومن ثم يمررها إلى الصنف لتعيينها إلى الكائنين المنشأين في البرنامج main من نوع الصنف time :

```
# include < iostream.h >
# include < iomanip.h >
class time
{
     private:
     int hour, minute;
     public:
     void set(int h, int m)
     {
        hour = h;
        minute = m;
     }
     void display( )
     {
        cout << hour << ':' << setfill ('0') << setw(2)
              << minute << endl;
     }
};
main ( )
{
     time t1, t2;
     int hh, mm;
     char ch;
     cout << " for t1, enter time (format 23:59):";
     cin >> hh >> ch >> mm;
     t1 . set (hh, mm);
     cout << " for t2, enter time (format 23:59):";
     cin >> hh >> ch >> mm;
     t2 . set (hh, mm);
     cout << "\n t1 = ";
     t1 . display ( );
     cout << "\n t2 = ";
     t2 . display( );
     return 0;
}
```

خرج البرنامج:

```
for t1, enter time (format 23:59): 10:04↵
for t2, enter time (format 23:50): 14:05↵
t1 = 10:04
t2 = 14:05
```

لقد ذكرنا سابقاً أن التابع setw(n) يحدد n عرض الحقل (عدد الرموز) وإذا لم يوجد قيم تستخدم الفراغات، أما التابع setfill فيحدد نوع الرمز المطلوب طباعته لو لم يكن هناك قيم لطباعتها وهو هنا 0 . نحتاج لتنفيذ التابعين السابقين إلى تضمين الملف الرأسي < iomanip.h > وتأثير التابع setw يدوم على القيمة المعروضة التالية فقط في تعليمة الطباعة بينما التابع setfill يدوم على كل القيم الواردة في تعليمة الطباعة.

11-2-9-3- التوابع الأعضاء التي تغير في قيم البيانات الأعضاء:

يمكن لأحد التوابع الأعضاء العامة في الصنف أن يقوم بتعديل قيم البيانات الأعضاء الخاصة في الصنف المثال التالي يوضح ذلك.

مثال (19) : يحوي الصنف time على تابع add( ) يقوم بتعديل أحد البيانات الخاصة في الصنف.

```
# include < iostream.h >
# include < iomanip.h >
class time
{
    private:
    int hour, minute;
    public:
    void set (int h, int m)
    {
      hour = h;
      minute = m;
    }
    void display( )
    {
      cout << hour << ':' << setfill ('0') << setw(2)
            << minute << endl;
    }
      void add(int h1)
    {
      hour = hour + h1;
      if (hour > 23)
          hour = hour – 24;
    }
};
```

```
main ( )
{
    int hh, mm, hd;
    char ch;
    time t1;
    cout << " for t1, enter time (format 23:59):";
    cin >> hh >> ch >> mm;
    t1 . set (hh, mm);
    cout << "enter hours to add:";
       cin >> hd;
       t1. add(hd);
       cout << " t1 = ";
       t1. display ( );
    return 0;
}
```

والخرج هو :

```
for t1, enter time (format 23:59): 22:45↵
enter hours to add:4↵
t1 = 2 :45
```

التابع add يضيف قيمة البارامتر hd إلى البيانات العضو hour وإذا كانت القيمة الناتجة أكبر من 23 عندها يطرح القيمة 24 منها ليحصل على الساعة الحقيقية.

11-2-9-4- التوابع الأعضاء العامة التي تجري عمليات حسابية على البيانات الأعضاء:

مثال (20) : البرنامج التالي يصرح عن صنف اسمه Exam له بيانات أعضاء خاصة كرقم الطالب stNO وعلاماته في ثلاثة مواد sb3mark, sb2mark, sblmark وتوابع أعضاء عامة واحدة لتعيين قيم للبيانات الأعضاء المدخلة في التابع main ومن ثم تمريرها له ، والتابع الآخر يحسب متوسط العلامات بالإضافة إلى متغير عام.

```
# include < iostream.h >
# include < iomanip.h >
class Exam
```

```
{
    private:
        int stdNO;
        double sblmark;
        double sb2mark;
        double sb3mark;
    public:
        double preAve;
        void insertstd (int, double, double, double);
        double calcavg( ) { return ( sblmark+sb2mark+sb3mark)/3; }
};
void Exam :: insertstd (int id, double math, double eng, double elect)
{
    stdNO = id;
    sblmark = math;
    sb2mark = eng;
    sb3mark = elect;
}
main ( )
{
    cout << setprecision (2)
            << setiosflags (ios :: fixed)
            << setiosflags (ios :: showpoint);
    int id;
    double math, eng, elect;
    cout << "\n enter student number:";
    cin >> id;
    cout << "\n enter Math mark :";
    cin >> math;
    cout << "\n enter English mark :";
    cin >> eng;
    cout << "\n enter Electronic mark :";
    cin >> elect;
    Exam st1;       // create an object st1
    st1.preAve = 65.00;     // assign 65.00 to public member data
    st1.insertstd (id, math, eng, elect);
    cout << "\n The average of these three marks is:"
           << st1. calcavg( );
    cout << "\n\n The value of previous Average is:"
           << st1. preAve;
    st1. preAve = (st1.calcavg( )+st1.preAve)/2;
    cout << " \n\n The new average is : " << st1.preAve;
    return 0;
}
```

والخرج هو :

```
enter student number : 12345↵
enter Math mark : 90↵
enter English mark : 80↵
enter Electronic mark : 70↵
The average of these three marks is : 80.00
The value of previous Average is : 65.00
The new average is : 72.50
```

11-2-9-5- إعادة قيمة من التوابع الأعضاء :

يمكن أن تعاد قيمة من التوابع الأعضاء لصنف ما، كما هو الحال في أي تابع درسناه سابقاً. ويمكن أن تكون تلك القيمة من أي نوع من أنواع البيانات المعروفة في لغة ++c . كما يمكن أن تكون من نوع صنف أيضاً .

مثال (21) : البرنامج التالي يقوم بتحويل الوقت إلى دقائق وطباعته.

```
# include < iostream.h >
# include < iomanip.h >
class time
 {
    private:
       int hour;
```

```
        int minute;
        public:
        void enter( );
        void display( );
        int TimeToMin( );
};
void time :: enter ( )
  {
    char ch;
    cout << " enter time (format 23:59):";
    cin >> hour >> ch >> minute;
  }
void time :: display( )
  {
    cout << hour << ':' << setfill ('0') << setw(2)
          << minute << endl;
  }
int time :: TimeToMin( )
  {
    int imin = hour * 60 + minute;
    return imin;
  }
main ( )
  {
    time t1;
    int m;
    cout << " for t1,";
    t1. enter( );
    cout << "t1 = ";
    t1. display( );
    m = t1. TimeToMin ( );
    cout << " minute = " << m << endl;
    return 0;
```

```
}
```

والخرج هو :

```
for t1, enter time (format 23:59): 11:15↵
t1 = 11:15
minute = 675
```

لقد تم الحصول على بيانات الكائن t1 في التابع main ثـم تم اسـتدعاء التـابع TimeToMin مع الكائن t1 ومرر له الزمن المدخل وقام هذا التابع بتحويل الزمن إلى دقائق وأعاده إلى التابع main حيث تم إظهاره.

مثال (22) : البرنامج التالي يستخدم تابعاً عضواً في صنف time يعيد قيمة مـن النـوع time :

```
# include < iostream.h >
# include < iomanip.h >
class time
 {
     private:
        int hour;
        int minute;
        public:
        void enter( );
        void display( );
        time add (time);
 };
 void time :: enter ( )
   {
     char ch;
     cout << " enter time (format 23:59):";
     cin >> hour >> ch >> minute;
    }
 void time :: display( )
```

```
  {
    cout << hour << ':' << setfill ('0') << setw(2)
          << minute << endl;
  }
time time :: add (time at2)
  {
    time temp;
    temp. minute = minute + at2. minute;
    temp. hour = hour + at2. hour;
    if (temp. minute > 59)
      {
        temp. minute = temp. minute-60;
        temp. hour = temp. hour + 1;
      }
    if (temp.hour > 23)
        temp.hour = temp.hour-24;
    return temp;
  }
main ( )
  {
    time t1, t2, t3;
    cout << " for t1,";
        t1. enter( );
    cout << "for t2, ";
        t2. enter( );
    cout << " t1= ";
        t1. display( );
    cout << "t2 = ";
        t2. display( );
    t3 = t1. add (t2);
    cout << " t3 = ";
        t3. display( );
    return 0;
}
```

والخرج هو :

```
for t1,enter time (format 23:59): 12:30↵
for t2, enter time (format 23:59): 14:45↵
t1 = 12:30
t2 = 14:45
t3 = 3:15
```

لقد تم استدعاء التابع ( )add للكائن t1 بتمرير وسيط له هو الكائن t2 . ولقد أنشأ التابع ( )add كائناً جديداً يدعى temp من نوع الصنف time وهو متغير محلي سيتم تدميره عند الخروج من التابع.

يخزن temp.minute وهو الزمن مقاساً بالدقائق والناتج من جمع دقائق minute (عضو البيانات للكائن t1) مع t2.minte (قيمة عضو البيانات minute للكائن t2)، وكذلك الأمر بالنسبة للساعات. والكائن temp يرجع إلى التابع ( )main في الكائن t3.

11-2-9-6- استدعاء التوابع الأعضاء بالمرجع

(Calling member function by reference)

عند تمرير البارامترات بالمرجع فإنه بالإمكان إرجاع قيم إلى البرنامج الطالب من دون وجود تعليمة العودة return في البرنامج المطلوب. ولكي نستطيع إعادة قيمه إلى البرنامج الرئيسي لايمكن أن يكون المتغير المرجعي معرفاً تحت القسم private في تعريف الصنف الموجود فيه هذا المتغير، لأن البيانات الأعضاء الخاصة في الصنف لايمكن الوصول إليها من خارج الصنف.

مثال (23): البرنامج التالي يعرف صنف fact لإيجاد عاملي عدد يمرر للتابع العضو بالمرجع.

```
# include < iostream.h >
# include < stdlib.h >
class fact
```

```
{
    private:
      int n;
    public :
    void start (int n1)
      {
        n = n1;
      }
    void factorial (int & f)
      {
        if (n < 0)
            {
              cout << " no factorial " << endl;
              exit(1);
            }
        f = 1;
        while (n>1)
            {
              f * = n;
              n--;
            }
      }
};
main ( )
{
    int n2 , f1;
    fact x;
    cout << " enter n to find n! :";
    cin >> n2;
    x. start (n2);
    x. factorial (f1);
    cout << n2 << " ! = " << f1 << endl;
    return 0;
}
```

خرج البرنامج هو :

```
enter n to find n! : 5↵
5! = 120
```

التابع العضو ( )start يخصص قيمة العدد المراد إيجاد عاملي له إلى عضو البيانات الخاص n. التابع ( ) factorial يقوم بإيجاد عاملي العدد n. قيمه !n تخزن في المتغير f الذي تم إرساله بالمرجع. تم الإعلان عن الكائن x من الصنف fact في التابع main( ) .

الاستدعاء :

```
x. start (n2);
```

سيحدد قيمة عضو البيانات n في الصنف بقيمة n2 . في حين أن الاستدعاء :

```
x. factorial (f1);
```

يحسب قيمة عاملي لعضو البيانات الخاصة n وتعاد قيمة العاملي في المتغير f1، ولكي نحصل عليها نقوم بعملية طباعة لتلك القيمة.

أما لو حدد بالإعلان أن f هو أحد البيانات الأعضاء الخاصة في الصنف fact. عندها لايمكن الوصول إلى f من خارج الصنف (أي من التابع ( )main) . ولابد لتأمين ذلك من كتابة تابع عضو يستخدم لطباعة قيمة العاملي التي خزنت في أحد البيانات الأعضاء وليكن f.

استخدم التابع exit(int n) والذي تَطَلب تضمين الملف الرأسي < stdlib.h > ويعمل هذا التابع على إيقاف البرنامج فور تنفيذه. وإذا كانت قيمة n تساوي الصفر فإن التابع سيوقف البرنامج، أية قيمة أخرى لـ n وهي في برنامجنا تساوي 1 يعني أن خطأ قد وقع.

10-2-11- البيانات الأعضاء الخاصة في الصنف (Data member in classes):

تعاملنا في برامجنا في هذا الفصل مع بيانات أعضاء خاصة بالصنف من الأنواع الأساسية كـ float, char, int . وسندرس الآن التعامل مع بيانات أعضاء أخرى من نوع مصفوفة array وحيدة البعد أو ثنائية البعد، وكذلك مع عضو بيانات مؤشر أو سلسلة string.

10-2-11-1- استخدام المصفوفة الأحادية كعضو بيانات في الصنف

(Array as data member in class)

نعلم أنه من غير الممكن تخصيص البيانات الأعضاء في الصنف سواء أكانت ثوابت أم لا . لذلك فإن التصريح عن مصفوفة أحادية ضمن الصنف بالشكل التالي غير مقبول:

```
private:
const int size = 20
int a [ size ];
```

ويكون الحل بالإعلان عن size خارج الصنف كثابت كما يلي :

```
# include < iostream.h >
const int size = 20;
:
:
private:
int a[size];
```

وهذا الخيار غير مجدٍ، لأن بيانات الصنف يجب أن تبقى خاصة بالصنف وليست عامة، بالإضافة لذلك فإنه لايمكن تخصيص قيم ابتدائية للمصفوفة داخل الصنف، كما هو متبع باستخدام قائمة التخصيص للقيم الابتدائية. والحل يكون بالإعلان عن المصفوفة كمتغير عام.

مثال (24) : البرنامج التالي يصرح عن صنف array له بيانات أعضاء خاصة أحدهما مصفوفة أحادية. وتوابع أعضاء عامة هي enter( ) لإدخال قيم لعناصر المصفوفة و print( ) لطباعة عناصر المصفوفة، و swap( ) لتبديل قيمتي عنصرين ، و sort( ) لترتيب عناصر المصفوفة ثم يصرح في التابع main( ) عن كائن من نوع الصنف array ويطلب إدخال قيم له ثم طباعته بعد الترتيب.

```
# include < iostream.h >
const int size = 20;
class array
{
     private:
        int a [size ];
        int n;
     public:
        void enter ( );
        void print ( );
        void swap (int&, int&);
        void sort( );
};
void array :: enter( )
{
     n = 0;
     for (int i = 0; i < size; i++)
        {
            cin >> a[i];
            if (a[i] = = 0)
                 break;
            n++;
        }
}
```

```
void array :: print ( )
{
    for (int i = 0; i < n; i++)
       cout << a[i] << "    ";
    cout << endl;
}
void array :: swap (int& x, int& y)
{
    if (x > y)
    {
       int temp = x;
       x = y;
       y = temp;
    }
}
void array :: sort( )
{
    for (int i=1 ; i<n; i++)
       for (int j=0; j<n-1; j++)
          if (a[j] > a[j+1])
               swap (a[j] , a[j+1]);
}
main ( )
{
    array x;
    cout << " enter array and 0 to end:";
    x. enter( );
    x. sort( );
    cout << " array after sort is:";
    x. print( );
    return 0;
}
```

الخرج هو :

```
enter array and 0 to end : 10  2  4  7  1  0↵
array after sort is : 1  2  4  7   10
```

الكائن x من الصنف array له بيانات أعضاء خاصة، الأول هو مصفوفة أحادية البعد a عدد عناصرها الأعظمي 20 ، عدد العناصر المدخلة يخزن في عضو البيانات الثاني n . لاحظ أن استدعاء التابع ( )swap يتم من داخل التابع ( )sort ، ويقوم بالتبديل ما بين قيمتي عنصرين متتاليين يمرران له بالمرجع إذا كان الأول أكبر من الثاني.

مثال (25): البرنامج التالي يستخدم مصفوفة ثنائية كعضو بيانات في الصنف array الذي يحوي على توابع أعضاء عامة هي ( )enter لإدخال قيم لعناصر المصفوفة و ( )print لطباعة تلك العناصر والتابع ( )summ لحساب مجموع هذه العناصر . ثم يصرح في التابع ( )main عن كائن من صنف array ويطلب إدخال قيم له وطباعتها وحساب مجموع تلك القيم وطباعته.

```
# include < iostream.h >
class array
{
    private:
       int a [3] [3];
       int sum;
    public:
       void enter ( );
       void print ( );
       int summ( );
};
void array :: enter( )
{
    for (int i=0 ; i<3; i++)
       for (int j=0; j<3; j++)
          cin >> a[i][j];
}
void array :: print( )
{
    for (int i=0 ; i<3; i++)
```

```
          for (int j=0; j<3; j++)
              cout << a[i][j] << "    ";
          cout << endl;
}
int array :: summ( )
{
      sum = 0;
      for (int i=0 ; i<3; i++)
          for (int j=0; j<3; j++)
              sum = sum + a[i][j];
      return sum;
}
main( )
{
      array x;
      int s;
      cout << " enter array:";
      x. enter( );
      cout << " array is:" << endl;
      x. print( );
      s = x. summ( );
      cout << " sum of all element is: " << s << endl;
      return 0;
}
```

خرج البرنامج هو :

```
enter array: 1  2  3  4  5  6  7  8  9↵
array is:
1    2    3
4    5    6
7    8    9
sum of all element is : 45
```

11-2-10-2- استخدام السلسلة كعضو بيانات في الصنف

(String as data member in class):

مثال (26): البرنامج التالي يصرح عن صنف str له بيانات أعضاء خاصة عبارة عن سلسلة محرفية وتوابع أعضاء عامة لإدخال السلسلة وطباعتها ثم يصرح في التابع main( ) عن كائنين x و y من الصنف str ويطلب إدخال قيم لهما ثم طباعتها.

```
# include < iostream.h >
class str
{
    private:
       char name [20];
       int serial;
    public:
       void enter_name( )
       {
           cout << " enter name :";
           cin. getline (name, 20);
           cout << " enter serial number:";
           cin >> serial;
           cin. ignore (1, '\n');
       }
void print( )
       {
           cout << " name = " << name << endl;
           cout << " serial number = " << serial << endl;
       }
};
main( )
{
    str x,y;
    x. enter_name( );
    y. enter_name( );
    x. print( );
    y. print( );
    return 0;
}
```

خرج البرنامج هو :

```
enter name: Mohamad Ali↵
enter serial number : 1↵
enter name : Samar Ali↵
enter serial number :2↵
name = Mohamad Ali
serial number = 1
name = Samar Ali
serial number = 2
```

تم تعريف الصنف str الذي له عضو بيانات هما السلسلة name عدد محارفها 19 محرفاً والمتغير الصحيح serial . كما أنها تملك تابعين عضوين هما enter_name( ) الذي يسمح بإدخال وتخزين اسم ، عدد حروفه أقل أو يساوي 19 من خلال التابع :

```
cin.getline(name, 20);
```

كما أنه يسمح بإدخال الرقم التسلسلي للاسم المدخل وتخزينه في المتغير serial.

أما التابع :

```
cin.ignor (1, '\n');
```

يتجاهل محرفاً من مجرى الدخل هو محرف سطر جديد '\n' . واستعماله ضروري، لأن التابع cin.getline( ) ينهي عمله بقراءة المحرف '\n' من مجرى الدخل، وهذا ما يحصل عند استدعاء التابع العضو enter_name( ) مرة ثانية من قبل الكائن y، حيث سيجد التابع cin.getline( ) المحرف '\n' في مجرى الدخل وينهي عمله من دون إدخال اسم وتخزينه في سلسلة ذلك الكائن، لذلك وجب استخدام التابع cin.ignore( ).

التابع العضو الثاني print( ) يسمح بطباعة الاسم والرقم التسلسلي على شاشة الحاسوب.

11-2-10-3- استخدام مصفوفة سلسلية كعضو بيانات في الصنف

(String array as data member in class)

مثال (27) : البرنامج التالي يصرح عن صنف أيام الأسبوع weekday له تابع عضو عام لطباعة أيام الأسبوع ويصرح عن كائن x من نوع الصنف السابق ويطبعه:

```
# include < iostream.h >
# include < string.h >
const int max = 10;
const int dpw = 7;
char day_name [dpw][max] = {"Sunday","Monday","Tuesday",
                             "Wednesday","Thursday","Friday",
                             "Saturday" };
class weekday
{
    public:
    void print( )
    {
       for (int i=0; i<dpw; i++)
           cout << i+1 << ":" << day_name[i] << endl;
    }
}:
main ( )
{
    weekday x;
    cout << " the days of week are : " << endl;
    x. print( );
    return 0;
}
```

خرج البرنامج هو :

```
the days of week are :
1 : Sunday
2 : Monday
3 : Ttuesday
4 : Wednesday
5 : Thursday
6 : Friday
7 : Saturday
```

نعيد ونؤكد أنه لايمكن إعطاء قيم ابتدائية لمصفوفة داخل الصنف وإنما خارجه.

ولكي نستطيع التعامل معها يجب أن يتم الإعلان عنها كمتغيرات عامة. الصنف weekday له تابع عضو عام هو print( ) الذي يسبب طباعة أيام الأسبوع.

11-2-10-4- استخدام المؤشر كعضو بيانات في الصنف

(pointer as data member in class):

مثال (28) : البرنامج التالي يوضح استخدام المؤشر كعضو بيانات ، حيث استعمل المؤشر في تحويل الأعداد الصحيحة الموجبة من النظام العشري إلى النظام الثماني.

نذكر أن أرقام النظام الثماني هي 0 , 1 ... 7 وأساسه 8 وعدد أرقامه 8.

أما أرقام النظام العشري هي 0 , 1 ...9 وأساسه 10 وعدد أرقامه 10 .

يصرح البرنامج أيضاً عن صنف اسمه DecToOct له التوابع الأعضاء set( ), pop( ), push( ) والبيانات الأعضاء وهي مؤشر p ، ومصفوفة صحيحة stack لقيمة صحيحة ، ومصفوفة ذات بعد أعظمي يساوي 50 عنصراً.

```
# include < iostream.h >
# include < stdlib.h >
const int stacksize = 50;
class DecToOct
```

```
{
     private:
        int * p;
        int stack [stacksize];
     public:
     void push (int i)
       {
        if (p = = (stack + stacksize))
         {
            cout << " stack over flow ";
            exit(1);
         }
        * p = i;
        p++;
       }
     int pop( )
     {
        if (p = = stack)
            {
                 cout << " stack under flow";
                      exit(1);
            }
            p--;
        return * p;
     }
     void set( )
     {
        p = stack;
     }
};
main( )
{
     DecToOct y;
```

```
    y. set( );
    int x, i, n = 0;
    do
    {
       cout << " enter positive number : ";
       cin >> x;
       if (x < 0)
           break;
       if (x = = 0)
           cout << "0";
       while (x! = 0)
           {
                i = x%8;
                y. push (i);
                n++;
                x = x/8;
           }
       while (n > 0)
           {
           cout << y. pop( ) << "    ";
           n--;
           }
       cout << endl;
    } while (x = = 0);
    return 0;
}
```

خرج البرنامج هو :

```
enter positive number : 9↵
1    1
enter positive number : 13↵
1    5
enter positive number : 102↵
1    4    6
```

```
enter positive number : 208
3   2   0
enter positive number : -1
```

التابع ()set يحدد قيمة للمؤشر بحيث يشير إلى بداية المصفوفة الأحادية stack .

أما التابع ( )push فيقوم بدفع (تخزين) عنصر في المصفوفة في الموقع الذي يشير إليه المؤشر p ، بعد أن يتحقق من عدم وجود طفحان (أي المصفوفة امتلأت بالعناصر)، ثم ينتقل المؤشر ليشير إلى الموقع التالي. أما إذا وجد حدوث طفحان فيطبع رسالة تدل على ذلك (باستخدام التابع (exit(1) ويتم الخروج من التابع نتيجة تنفيذه هذا التابع المعرف في الملف الرأسي < stdlib.h >.

التابع العضو ( )pob يستخدم لسحب عنصر من المصفوفة.

لاحظ أن آخر تعليمة في التابع ( )push هي تعليمة زيادة قيمة المؤشر ليشير إلى موقع غير مخصص جديد، أما التابع ( )pop فعليه إنقاص قيمة المؤشر أولاً ليصل إلى عنصر المصفوفة لإعادته إلى البرنامج الرئيسي بالتعليمة ;return * p ، وذلك بعد التأكد من عدم وجود حالة جفاف (أي عدم وجود أي عنصر في المصفوفة لسحبه).

11-2-11- التوابع الصديقة (friend function):

لقد ذكرنا أن خصائص الصنف هي خاصة (private) ولايمكن الوصول إليها إلاّ من قبل التوابع المعرفة معها في الصنف. ولكن في بعض الحالات قد نحتاج إلى الوصول إلى بعض البيانات الخاصة لصنف معين من خلال تابع موجود خارج الصنف، عندها نعرف هذا التابع كتابع صديق. والتوابع الصديقة لصنف معين تكون عادة خارج الصنف ولكنها تتمتع بمزايا التوابع الموجودة في الصنف نفسه، حيث يمكنها الوصول إلى جميع الصفات الخاصة والمعرفة في الصنف. وتجدر الإشارة إلى أن الصنف يجب أن يتضمن تصريحاً عن التابع الصديق وهذا مهم جداً، ويتم ذلك بإضافة الكلمة المفتاحية friend قبل نموذج تعريف التابع ضمن تعريف الصنف وقبل نوع البيانات الذي يرجعه

ولايهم موضع التصريح ضمن الصنف وضمن المنطقة private أو public أو خارجهما كما هو مبين فيما يلي:

```
class class_name
{
     friend return_data_type function_name (arguments data types);
     private:
     .........
     public:
     .........
};
```

مثال (29): يبين البرنامج التالي كيفية التصريح واستخدام التابع الصديق setx من أجل إعطاء قيمة لمتحول المعطيات العضو الخاص x في الصنف const.

```
// friends can access private members of a class
# include < iostream.h >
// modified count class
class count
  {
     friend void setx (count&, int);      // friend declaration
     public:
        count () { x = 0; }                      // constructor
        void print () const { cout << x << endl; }           // output
     private:
        int x;          // data member
  };
// can modify private data of count because setx
// is declared as a friend function of count
void setx( count & c, int val)
     {
        c. x = val;          // legal: setx is a friend of count
     }
main()
```

```
{
     count object;
     cout << " object. x after instantiation:";
     object. print();
     cout << " object. x after call to setx friend function:";
     setx (object, 8);          // set x with a friend
     object. print();
     return 0;
}
```

والخرج هو :

```
object. x after instantiation: 0
object. x after call to setx friend function: 8
```

مثال (30): يبين البرنامج التالي أنه لايمكن استدعاء تابع غير صديق cannotSetx( ) لتعديل قيمة المتحول x.

```
// Non-friend / non_member function cannot access
// private data of a class
# include < iostream.h >
// modified count class
class count
     {
        public:
            count() { x = 0; }              // constructor
            void print() const {cout << x << endl; }        // output
        private:
            int x;                  // data member
     };
// function tries to modify private data of count,
// but cannot because it is not a friend of count.
void cannotSetx (count& c, int val)
     {
        c.x=val;          // Error : 'count.x' is not accessible
     }
```

```
main()
{
    count object;
    cannotSetx(object, 3);          // cannotSetx is not a friend
    return 0;
}
```

والخرج هو رسالة تعلمنا أنه لايمكن الوصول إلى المتغير x من خلال التابع غير الصديق () cannotSetx.

12-2-11- قالب التابع (Function Template):

تستخدم زيادة التحميل في التوابع عادة للقيام بعمليات متشابهة على بيانات من أنواع مختلفة. لكن إذا كانت هذه العمليات متطابقة من أجل كافة أنواع المعطيات المستخدمة، فإنه يمكن تحقيق ذلك بشكل أفضل وأكثر ملاءمة باستخدام إمكانية بناء قوالب التوابع (Function Templates) التي أضيفت إلى لغة ++C حيث يكتب المبرمج تعريفاً وحيداً لقالب تابع ما . ثم تقوم ++C بتوليد توابع منفصلة لمعالجة كل نوع من أنواع البيانات المستخدمة عند استدعاء التابع. أي أن تعريف قالب وحيد لتابع يعطي إمكانية للحصول على مجموعة من التوابع.

تبدأ تعريفات جميع قوالب التوابع بالكلمة المفتاحية template التي يرد بعدها قائمة بأنواع بارامترات التابع الوهمية بين الإشارتين > و < . يُسبق كل نوع للبارامتر الكلمة class. يمكن أن تكون أنواع البارامترات الوهمية إحدى الأنواع المعرفة مسبقاً في لغة ++C أو أي نوع من الأنواع المعرفة من قبل المستخدم ويتم استخدام هذه الأنواع لتحديد أنواع بارامترات التابع أو نوع القيمة المعادة أو حتى من أجل التصريح عن المتحولات المستخدمة داخله. يتم بعدها المتابعة في تعريف جسم القالب كما لو كنا نقوم بتعريف جسم أي تابع آخر.

التعريف التالي لقالب أحد التوابع maximum

```
template < class T >
```

```
T maximum (T value1, T value2, T value3)
{
        T max = value1;
        if (value2 > max)
            max = value2;
        if (value3 > max)
            max = value3;
        return max;
}
```

يصرح قالب التابع السابق عن باراميتر وهمي من النوع T، وبالتالي عندما يتحسس المترجم أي استدعاء للتابع maximum ضمن نص البرنامج فإنه يقوم باستبدال النوع T بنوع البيانات المستخدمة أثناء عملية الاستدعاء وذلك من أجل خلق نسخة كاملة عن التابع لحساب القيمة العظمى من بين ثلاثة قيم من النوع المحدد. يتم بعدها ترجمة نسخة التابع التي تم خلقها. بناءً على ذلك، نستطيع أن نعتبر القوالب كما لو أنها عبارة عن مولدات للتوابع. نجد في المثال (29) ثلاث طرائق لاستدعاء التابع maximum مع الأنواع : int و double و char ، مما سيؤدي إلى خلق ثلاثة نسخ عن هذا التابع من أجل كل نوع . سوف نأخذ النسخة المتعلقة بالنوع int الشكل التالي:

```
int maximum (int value1, int value2, int value3)
{
    int max = value1;
    if (value2 > max)
        max = value2;
    if (value3 > max)
        max = value3;
    return max;
}
```

يجب الانتباه إلى وجوب ورود كل باراميتر وهمي مرة واحدة على الأقل ضمن قائمة البارامترات. كذلك، يجب أن يكون اسم الباراميتر الوهمي وحيداً ضمن قائمة البارامترات المتعلقة بتعريف تابع ما.

مثال (31) : البرنامج التالي يصرح عن قالب التابع maximum ثم يقوم باستدعائه ثلاثة مرات من أجل أنواع للبارامترات char, double, int .

```
// using template functions
# include < iostream.h >
template < class T >
T maximum (T value1, T value2, T value3)
{
    T max = value1;
    if (value2 > max)
       max = value2;
    if (value3 > max)
       max = value3;
    return max;
}
main()
{
    int int1, int2, int3;
    cout << " Input three integer values:";
    cin >> int1 >> int2 >> int3;
    cout << " The maximum integer value is:"
         << maximum (int1, int2, int3);          // int version
    double double1, double2, double3;
    cout << endl << "Input three double values:";
    cin >> double1 >> double2 >> double3;
    cout << " The maximum double value is:"
         << maximum (double1, double2, double3);   // double version
    char char1, char2, char3;
    cout << endl << " Input three characters:";
    cin >> char1 >> char2 >> char3;
    cout << " The maximum character value is:"
         << maximum (char1, char2, char3) << endl;   //  char version
    return 0;
}
```

والخرج هو :

```
Input three integer values: 1  7  4⏎
The maximum integer value is:  7
Input three double values:  4.4   2.1   5.6⏎
The maximum double value is: 5.6
Input three character values: B  F  I⏎
The maximum character value is: I
```

13-2-11- التوارث (Inheritance) :

لقد عرفنا انتقال الصفات الوراثية من كروموسومات الآباء إلى الأبناء وهذه العملية تسمى بالوراثة ولها قوانينها الخاصة. أما الوراثة بالنسبة للبرمجة وكما ذكرنا سابقاً فهي الاستفادة من بعض الصفات في برنامج دون إعادة كتابة تلك الصفات. ويقصد بالصفات التوابع أو المتغيرات التي تؤدي دوراً معيناً، توفر لغة ++C وسيلة جيدة لتقليل حجم البرامج حيث يتم الاستفادة من بعض التوابع في بعض الأصناف وهذا يعني اشتقاق أصناف جديدة من أصناف محددة وذلك بأخذ التوابع المشتركة أو التي تؤدي نفس الغرض ثم إضافة المواصفات الإضافية وبهذه الطريقة لاتحتاج إلى إعادة كتابة التوابع في كل مرة. ومن أجل تقريب مفهوم التوارث أكثر لابدّ من الاستعانة بالأمثلة التالية:

مثال (32) : التعليمات التالية تعرف صنفاً rectangular لتحديد طول وعرض مستطيل:

```
class rectangular
{
    public:
        float width, length;
        void get_width (float w)
            {
                width = w;
            }
        void get_length (float l)
            {
                length = l;
            }
};
```

مثال (33) : التعليمات التالية تعرف صنفاً color_rectangular لتحديد طول وعرض مستطيل ثم تلوين ذلك المستطيل.

```
class color_rectangular
{
    public:
    float width, length;
    void get_width (float w)
        {
            width = w;
        }
    void get_length (float l)
        {
            length = l;
        }
    int color;
    void get_color (int c)
        {
            color = c;
        }
};
```

نلاحظ أن المثال (33) يحتوي على ثلاثة متغيرات وثلاثة توابع، حيث التابعان get_width و get_length قد تم تعريفهما في صنف سابق كما في المثال (32). والسؤال هنا هل يمكن الاستفادة من الصنف في المثال (32) دون إعادة كتابة التابعين مرة أخرى من جديد، والجواب نعم يمكن الاستفادة من ذلك وهذا هو ما يسمى بالتوارث أو أخذ بعض المواصفات في صنف وإضافة مواصفات جديدة لها في صنف آخر أو حذف قسم من المواصفات وهذا ما سنبينه في المثال التالي :

مثال (34) : بما أن المثال (32) يحتوي على متغيرين هما width و length وتابعين get_width و get_length وهذان المتغيران وهذان التابعان استخدما في المثال (33). لذا نستطيع الاستفادة من مزايا لغة ++C لإعادة كتابة المثال (33) دون إعادة كتابة التابعين والمتغيرين في المثال (32) وكما يلي:

الصنف الوارث ← color_rectangular

الصنف الأساس ← rectangular

```
class   color_rectangular  :  public  rectangular
   {
       public:
          int color;
          void get_color (int c)
              {
                   color = c;
              }
   };
```

العناصر الإضافية ← int color; void get_color (int c)

يسمى الصنف rectangular بصنف الأساس (الأب) ويسمى الصنف color_rectangular بالصنف المشتق (derived class) أو الصنف الوارث.

اشتقاق صنف من صنف أساس يذكر اسم الصنف الأساس بعد الصنف الوارث وكما يلي:

الكلمة المحجوزة class أو struct ← < class_specifier >

الصف الوارث ← < derived_class_name >

الصف الأساس ← < base_class_name >

```
< class_specifier > < derived_class_name > : < base_class_name >
   {
      < extra members >
   };
```

العناصر الإضافية ← < extra members >

حيث :

class_specifier : عبارة عن إحدى الكلمتين المحجوزتين class أو struct علماً أنه لايمكن اشتقاق أصناف جديدة من الاتحادات unions.

المثال التالي يوضح البنية struct وطرائق اشتقاقها.

مثال (35) : اكتب تعليمات تعريف الصنف التي توصف نقطة p(x,y) ثم استخدم ذلك الوصف لتحديد مساحة دائرة.

نعرف أولاً النقطة p(x,y) بالصنف p :

```
class  p
{
    float x,y;
    void get_points (float first, float second)
        {
            x = first;
            y = second;
        }
};
```

نعرف ثانياً صنف الدائرة كما يلي :

```
class  circle : p       // circle inherits p
    {
        // additional members
        float radius;
        void get_radius (float r)
            {
                radius = r;
            }
    };
```

وهكذا نرى أن الصنف الجديد أو الوارث circle قد استفاد من الصنف الأساس p حيث تم إضافة عناصر جديدة للصنف الوارث وبذلك أصبحت عناصر الصنف الجديد هي عناصر الصنف الأساس مضافاً إليها العناصر الإضافية. والعناصر

الموروثة من الصنف الأساس يمكن الرجوع إليها مباشرة وكأنها معرفة ضمن الصنف الوارث. وبعبارة أخرى يمكنك الاستفادة واستخدام العناصر x و y و get_point() مباشرة وكما هو مبين أدناه :

```
circle  small;
```

التصريح عن كائن اسمه small من الصنف circle

```
small. x = 120.0;
small. y = 80.0;
```

قيم x و y المعرفة في الصنف p هي 120.0 , 80.0

```
small.get_points (20.0, 30.0);
```

استدعاء التابع get_ points() المعرف في الصنف p

```
small. radius = 10.0;
small. get_radius (21.0);
```

كل من المتغير radius والتابع get_radius() معرفان في الصنف circle

وهكذا نرى أن الإشارة إلى العناصر الموروثة من صنف آخر تعامل وكأنها عناصر معرفة في الصنف الوارث. والآن سنكتب البرنامج الكامل وكما يلي :

```
// circle.h
struct  p
{
    float x,y;
    void get_points (float first, float second)
      {
         x = first;
         y = second;
      }
};
struct  circle : p
    {
      float radius;
```

```
          void get_radius (float r)
              {
                   radius = r;
              }
     };
// main program
# include < iostream.h >
# include " circle.h "
main()
{
     circle small;        // declare an object small of type circle
                         // which is defiend in the file circle.h
     small.get_points(20.0, 30.0);
     small.get_radius(21.0);
     cout << small.x << " \t " << small.y << " \t " << small. radius;
}
```

وكمثال آخر على توارث الأصناف class inheritance ندرس برنامج العناوين البريدية. لنفرض أن المبرمج احتاج في مرة سابقة إلى كتابة برنامج يستخدم صنفاً لتحديد الاسم الثلاثي لشخص ما. وفي مرة أخرى احتاج أن يحدد صنفاً لإيجاد كيان يحوي عنوان شخص ما. من الواضح أن العنوان يحتوي الاسم في داخل صنف العنوان وأنه بالإمكان استخدام صنف الاسم السابق.

مثال (36): لاحظ البرنامج الآتي :

```
# include <iostream.h>
# include <string.h>
class name
 {
  private:
    char first [1];
    char mid [1];
    char last [1];
  public:
     name() {first [1] = mid[1] = last [1] = '   ';}
     void set_name (char*, char*, char*);
     char* show ();
};
void name: : set_name (char* f, char*m, char*l)
 {
   strcpy (first, f);
   strcpy (mid, m);
   srcpy (last, l);
 }
char* name: : show()
 {
    char* temp = new char [20];
    strcpy (temp, first);
    strcat (temp, "  ");   // add a space
    strcat (temp, mid);
    strcat (temp, "    ");
    strcat (temp, last);
    return temp;
}
class address : name
 {
    private:
       char street [1];
       char city [1];
       char state [1];
       char zip [1];
    public:
     address() {street [1] = city [1] = state [1] = zip[1]=' ';}
     void new_address (char*, char*, char*, char*);
     void new_name (char*, char*, char*);
     char* show_info();
```

```
};
void address : : new_address (char* s, char* c, char* st,char*z)
 {
   strcpy (street, s);
   strcpy (city, c);
   strcpy (state, st);
   strcpy (zip, z);
 }
void address : : new_name (char* f, char* m, char* l)
 {
  set_name (f, m, l);
  }
char* address : : show_info ()
{
  char* temp = show();
  strcat (temp, "\n");
  strcat (temp, street);
  strcat (temp, "\n");
  strcat (temp, city);
  strcat (temp, " , ");
  strcat (temp, state);
  strcat (temp, "   ");
  strcat (temp, zip);
  return temp;
}
main()
 {
   address x;
   x.new_address("1234 khalid Street", "Homes", "Syria","94080");
   x.new_name ("saleem", "salman", "noor");
   cout << "**" << x. show_info() << "\n";
   return 0;
 }
```

يبدأ هذا البرنامج بتعريف الصنف class name . هذا الصنف يحتوي ثلاثة عناصر خاصة هي الاسم الأول والاسم الأوسط والاسم الأخير. كما أنه يحتوي على ثلاث توابع في القسم العام هي : التابع ()name وينفذ هذا التابع بشكل تلقائي عند تعريف كيان يستخدم الصنف name. أما التابعين ()set_name و()show فإنهما يستخدمان لإعطاء قيم للعناصر الخاصة من الصنف أو لطباعة هذه العناصر.

نحن نفترض أن الحاجة إلى هذا الصنف name كانت سابقة وجاءت حاجة جديدة لإضافة الصنف address. وكما سبق وأن ذكرنا فإن الصنف address يحتاج إلى عناصر كمثل تلك المذكورة في الصنف name. ولذلك فإننا نقوم باستعمال الصنف name كقاعدة للصنف address ويسمى الصنف address بالصنف المشتق. أو (derived class) . وحيث إن الصنف address يحتاج لعناصر أخرى غير مذكورة في الصنف name فإن هذه العناصر تدرج في تعريف الصنف address. وبذلك فإن الصنف address قد ورث عناصر الصنف name وأضاف إليها عناصر أخرى مثل state, city, street و zip. تلاحظ أن عملية وراثة مكونات الصنف name من قبل الصنف address قد جاءت على شكل class address: name.

وللاستفادة من اشتقاق الأصناف يجب الانتباه الشديد إلى قواعد هذا الاشتقاق فمثلاً نقول إن الصنف address قد ورث عناصر الصنف name. غير أنه يجدر الانتباه إلى أن الكيان address لايملك استخدام أي من العناصر الموجودة في القسم الخاص من الصنف name بشكل مباشر وهذا قد يبدو محيراً للوهلة الأولى. غير أن القاعدة تقول إن الصنف address يمكنه الوصول إلى عناصر الصنف name من خلال دوال القسم العام في الصنف name وهذا يمنع أي استخدام غير طبيعي لعناصر القسم الخاص (ليس شرطاً أن يحوي القسم العام توابع فقط، بل يمكن أن يحوي على متغيرات أيضاً) لاحظ كيف تم استخدام التابع ()new-name في الصنف address. لقد كان الغرض من هذا التابع هو إعطاء قيمة للاسم. أو بعبارة أخرى إعطاء قيم لعناصر الصنف

name. لقد استخدم التابع new-name() التابع set-name الموجود في القسم العام من الصنف name للقيام بهذه المهمة.

وفي المقابل تلاحظ كيف تعامل التابع new-address() مع المتغيرات street، city ، state و zip . لقد امتلك هذا التابع إمكانية إعطاء قيم لهذه المتغيرات بشكل مباشر وذلك لأن هذه المتغيرات كانت مدرجة في القسم الخاص من الصنف address ولم تكن مورثة من صنف آخر.

إن توابع ومتغيرات القسم العام في الصنف name هي في متناول استخدام توابع الصنف المشتق address وذلك كما هو ظاهر من خلال التابع (new-name) الذي استخدم التابع (set-name) من الصنف name . ولكن ماذا عن كيان من نوع الصنف address كالكيان ;address my-home.

هل يستطيع الكيان my-home استخدام أي من توابع الصنف الموروث name؟ الجواب هو النفي. لتوضيح هذه النقطة لاحظ المثال الآتي:

```
my-home. show-info();
```

تقوم هذه التعليمة بطباعة عنوان الكيان my-home . وقد توفرت هذه الإمكانية لأن show-info() هي إحدى توابع القسم العام في الصنف address الذي هو صنف الكيان my-home غير أن التعليمة my-home. show() التي يتوقع منها أن تقوم بطباعة الاسم فقط هي غير صحيحة وذلك لأن التابع show() هو أحد توابع الصنف الموروث name وعلى الرغم من أن هذا التابع موجود في القسم العام من هذا الصنف.

قد يبدو أن عملية التقييد هذه جاءت غير ضرورية ولكن هناك حالات يكون فيها هذا التقييد لازماً وأكثر أماناً. أما إذا اختار المبرمج أن يجعل القسم العام من الصنف الموروث قابلاً للتعامل كأنه قسم عام في الصنف المشتق (الوارث) فإن لغة C++ توفر هذه الإمكانية كما يلي :

عند تعريف الصنف address تضاف كلمة public قبل اسم الصنف المـوروث name أي name address:public class عنـــدها فـــإن اســـتخداماً مثـــل my-name.show() يكون صحيحاً.

ندرس مثالاً آخر على الأصناف الموروثة :

مثال (37) :

```
# include <iostream. h>
# include <string. h>
class student
 {
   private:
     char first [80];
     char mid [80];
     char last [80];
     char id [10];
   public:
      student (char*, char*, char*, char*);
      void show_student();
 };
student : : student (char* f, char* m, char* 1, char* i)
{
  strcpy (first, f);
  strcpy (mid, m);
  strcpy (last, 1);
  strcpy (id, i);
}
void student : : show_student()
{
  cout << "Name: \n";
  cout << "\t" << first << "   "<< mid << "   " << last << "\n";
  cout << "Identification: \n";
  cout << "\t" << id << "\n";
}
class course: public student
{
  private:
    char c_id [10];
    char description [128];
    char grade;
  public:
     course (char*, char*, char*, char*, char*, char*);
     void set_grade (char g) {grade = g;}
     void show_grade();
};
course : : course (char* f, char* m, char* 1, char* si, char* ci, char* cd) :
              student (f, m, l, si)
{
  strcpy (c_id, ci);
  srtrcpy (description, cd);
}
void course : : show_grade()
{
  cout << "Course identification: " << c_id << "\n";
  cout << "= = = = = = = = = = = \n";
  cout << description << "\n";
}
main()
{
  course x ("Ahmad", "Salma", "Zeinab", "012349876",
             "CIS 25c", "programming in C++");
  x. set_grade ('A');
  x. show_student ();   // "borrowed" from the base class
  x. show_grade();
  return 0;
}
```

في هذا المثال الصنف student هو الصنف الموروث والصنف course هو الصنف الوارث . عند تعريف الصنف course قمنا بإضافة كلمة public قبل student كما يلي class course : public student وبذلك فإن الكيان x المعرف بـ ..course x يستطيع استخدام الجزء العام من الصنف course كما في ;('x. set_grade ('A كما أنه يستطيع استخدام الجزء العام في الصنف الموروث student كما في .x ;()show_student نلاحظ أنه بإمكاننا التراخي في إخفاء الجزء العام من الصنف الموروث حسب حاجتنا وذلك كما أسلفنا. والآن يأتي السؤال ماذا عن الجزء الخاص بالصنف الموروث وهل يمكن أن تتوفر إمكانية لتوابع الصنف المشتق كي تتعامل معه، لغاية الآن الجواب هو النفي. أي أن القسم الخاص يبقى مستتراً حتى عن توابع الصنف الوارث. وقد لايعجب هذا التقييد أيضاً بعض المبرمجين . ولذلك فإن لغة ++C أيضاً قد منحت المبرمج أن يرفع هذا القيد وحسب رغبته وذلك يتم كما يلي :

لنفرض أن المبرمج أراد أن يجعل العنصر id من الصنف student قابلاً للتصرف به من قبل توابع الصنف course. عندها يجب إعادة كتابة الصنف student كما يلي:

```
class student
{
  private:
    char first [80];
    char mid [80];
    char last [80];
  protected:
    char id[10];
  public:
     student (char*, char*, char*, char*);
   void show_student();
};
```

لاحظ استخدام الكلمة الجديدة protected . إن المتغير id أصبح في نطاق هذا القسم الجديد فهي ليست ضمن القسم الخاص ولا القسم العام. فالمتغير id لايزال مستتراً في البرنامج ولكنه أصبح الآن ظاهراً لدى توابع الأصناف المشتقة (الوراثة).

لقد كان بإمكان المبرمج أن يضع المتغير id في القسم العام وبذلك يكون متوفراً لاستخدام توابع الأصناف المشتقة. ولكنه في تلك الحالة سيكون ظاهراً ومتوفراً لجميع الكيانات الأخرى وهذا قد لايكون مرغوباً في بعض الأحيان.

نلاحظ هنا مرة أخرى أن هناك رغبة في إخفاء بعض الأجزاء من الأصناف باستمرار وإذا اضطررنا لكشفها لحاجة ما فإننا نقوم بإخفاء هذه العناصر عن أي استخدام خارج هذه الحاجة. وهذا أساس في برمجة الكيانات لأننا نريد أن يبقى استخدام عناصر الكيان فقط من خلال الطرق الصحيحة وإخفاء العناصر يحميها من أي خطأ غير مقصود.

11-3- تمارين:

1- اكتب برنامجاً يصرح عن صنف Rational له تابع بانٍ و تابع باني ناسخ وتابع آخر للطباعة وجميعها توابع أعضاء عامة، وله بيانات أعضاء خاصة وتابعين أعضاء خاصين كذلك. الأول gcd( ) يعيد قيمة القاسم المشترك الأعظم لعددين صحيحين مررا له، والتابع reduce ( ) يستخدم التابع gcd لاختزال الكسر إلى أقل درجة. بحيث يختصر الكسر 100/360 إلى 5/18.

```
# include < iostream.h >
class Rational
{
    public:
    Rational (int n = 0, int d = 1): num(n), den(d) { reduce ( ); }
    Rational (const Rational & r): num(r.num), den(r.den) { }
    void print( ) { cout << num << '/' << den;}
    private:
     int num, den;
     int gcd (int m, int n)
     {
        if (n = = 0)
        return m;
        return gcd (n, m%n);
     }
    void reduce( )
     {
        int g = gcd (num, den);
        num / = g;
        den / = g;
     }
};
main ( )
{
     Rational x(100, 360);
     Rational y(x);       // copy object x into object y
     cout << " x = ";
     x. print( );
     cout << " , y = ";
     y.print( );
}
```

والخرج هو :

```
x = 5/18, y = 5/18
```

2- اكتب برنامجاً يعرف صنفاً fact لايجاد عاملي عدد و يصرح عن عضو بيانات f بأنه خاص في الصنف fact. و تتم طباعة العاملي المخزن في f عن طريق تابع عضو في الصنف fact نفسه .

```
# include < iostream.h >
# include < stdlib.h >
class fact
{
     private:
        int n, f;
     public:
        void start (int n1, int f1)
        {
            n = n1;
            f = f1;
        }
     void factorial ( )
        {
            if (n < 0)
            {
                 cout << " no factorial." << endl;
                 exit(1);
            }
            int m = n;
            while (m > 1)
              {
                 f * = m;
                 m--;
              }
        }
     void print( )
        {
            cout << n << " ! = " << f << endl;
        }
};
main( )
{
     int n2;
     fact x;
     cout << " enter n to find n! :";
     cin >> n2;
```

```
    x. start (n2,1);
    x. factorial ( );
    x. print( )
    return 0;
}
```

تم أخذ نسخة من n في المتغير m لنحافظ على قيمة n لتستخدم في التابع العضو print( ) بشكل صحيح.

تم في الاستدعاء ;x. start (n2,1) تخصيص القيمة 1 لعضو البيانات f وهي القيمة الابتدائية اللازمة.

3- اكتب برنامجاً يقوم بإنشاء ومعالجة كيان من الصنف Inventory ، بياناته الخاصة هي: قيمتين محرفيتين الأولى رقم المادة ItNo والثانية اسمها ItName وقيمتين حقيقيتين الأولى تمثل الكمية المتوفرة qtyOnHand والثانية تمثل الكمية المحجوزة qtyReserved وعدة توابع عامة بحيث يستطيع تنفيذ عملية الإضافة Addition() وعملية السحب Withdrawal() من المخزون ويحدث خصائص الكيان بعد كل عملية ويظهر قيمها على الشاشة dispItNo() و dispItName().

```
# include < iostream.h >
# include < iomanip.h >
# include < string.h >
class Inventory
{
    private:
        char ItNo[5];
        char * ItName;
        double qtyOnHand;
        double qtyReserved;
    public:
       Inventory (char id[ ], char* nam, double aqty = 20.0,
                     double rqty = 10.0);
    ~ Inventory();
       double getAvailableQty();
       double getQtyOnHand();
       void Addition (double);
       int withdrawal (double);
       void dispItNo();
       void dispItName();
    };
Inventory : : Inventory (char id[ ], char*nam, double aqty, double rqty)
    {
       strcpy (ItNo,id);    // copy first argument into ItNo[ ]
       ItName=new char[strlen(nam)+1];   //create space for the name
       strcpy (ItName,nam);  // copy second argument into new space
       qtyOnHand = aqty;
       qtyReserved = rqty;
    }
Inventory : : ~ Inventory()
    {
       cout << " \n\n Inventory " << ItNo<< " terminated";
       delete [ ] ItName;
    }
inline double Inventory : : getQtyOnHand()
    {
       return qtyOnHand;
    }
double Inventory : : getAvailableQty()
    {
       double availQty;
       availQty = qtyOnHand-qtyReserved;
       return availQty;
    }
void Inventory : : Addition (double amount)
    {
      qtyOnHand + = amount;
    }
int Inventory : : withdrawal (double amount)
    {
      int Flag;
      if (amount < = qtyOnHand)
          {
              qtyOnHand = amount;
              Flag = 1;
          }
      else
              Flag = 0;
      return Flag;
    }
void Inventory : : dispItNo()
    {
      cout << ItNo;
    }
void Inventory : : dispItName()
    {
      cout << ItName;
    }
main()
    {
      cout << setprecision(2) << setiosflags(ios : : fixed)
            << setiosflags(ios : : showpoint);
      char id [6];
      char buffer [70];    // Temporarily stores the name
      double aqty;
      double rqty;
      double amount;
    cout << "\n Enter Inventory ID:";
    cin.getline (id, 6);
    cout << "\n Enter Inventory Item Name: ";
    cin.getline (buffer, 70);
    cout << "\n Enter Quantity On Hand: ";
    cin >> aqty;
    cout << "\n Enter Reserved Quantity: ";
    cin >> rqty;
    Inventory item1 (id, buffer, aqty, rqty);
    cout << " \n The Inventory Item number is: ";
    item1.dispItNo();
    cout << "\n The Inventory Item Name is: ";
    item1. dispItName();
    cout << "\n\ The Quantity On Hand is: "
          << item1.getQtyOnHand();
    cout << "\n\n Enter a Quantity to add: ";
    cin >> amount;
    item1.Addition (amount);
    cout << "\n\ An Addition of  "<< amount << " was made";
    cout << "\n The Quantity on Hand is now  "
           << item1.getQtyOnHand();
    cout << "\n The Available Quantity is now  "
           << item1.getAvailableQty();
    cout << " \n\n Enter the Quantity to withdraw: ";
    cin >> amount;
    if (item1.withdrawal (amount))
        cout << " \n\ A withdrawal of "<< amount << " was made.";
    else
        cout << " \n\ Insufficient Inventor";
    cout << " \n The quantity on hand is now "
           << item1.getQtyOnHand();
    return 0;
}
```

والخرج هو :

```
Enter Inventory ID : 1417↵
Enter Inventory Item Name : Personal computer↵
Enter Quantity On Hand : 25↵
Enter Reserved Quantity : 15↵
The Inventory Item number is : 1417
The Inventory Item Name is Personal computer
The Quantity On Hand is 25.00

Enter a Quantity to add : 25↵
An Addition of 15.00 was made
The Quantity on Hand is now 50.00
The Available Quantity is now 35.00

Enter the Quantity to withdraw : 15↵
A withdrawal of 15.00 was made.
The quantity on hand is now 35
Inventory 1417 terminated.
```

4- اكتب برنامجاً يصرح عن الصنف Inventory وإنشاء الكيان item1 المنتمي له ومعالجته. يتضمن هذا الصنف البيانات الأعضاء التالية : قيمة صحيحة تمثل رقم المادة itemId . وقيمتين حقيقيتين الأولى تمثل الكمية المتوفرة من المادة qtyonHand والثانية تمثل الكمية المحجوزة qtyReserved ، وعدة توابع منها تابع بناء الكيان Inventory() وتابع قراءة الكمية المتوفرة getQtyOnHand() وتابع حساب الكمية المتاحة من المادة getAvailableQty() وتسجيل عمليات الإضافة إلى المخزون Addition() والسحب من المخزون Withdrawal().

5- أعد كتابة البرنامج السابق واستخدم تابع البناء Inventory() لإيجاد الكيان Item1 للصنف Inventory الموصوف في التمرين السابق ثم قم بهدمه أو إزالته بالتابع الهادم ~Inventory().

6- اكتب برنامجاً يعرف قالباً للتابع swap() وقالباً للتابع max() وقالباً للتابع sum() ثم استدع هذه التوابع من أجل بارامبترات ثنائية ذات أنواع int أو double أو char.

7- اكتب برنامجاً يعرف صنفاً اسمه account يتضمن ثلاث خصائص هي رقم الحساب accNo واسم الحساب accName والرصيد accBalance . ثم يطلب من المستخدم إدخال البيانات لإنشاء كائنين (حسابين) هما acc1 و acc2 ثم يستخدم تابعاً صديقاً ليقوم بتحويل المبالغ بين كائنات الصنف من حساب لآخر. لهذا التابع الصديق ثلاث بارامبترات هي رقم الحساب في الكائن المعطي ورقم الحساب في الكائن المتلقي والمبلغ المحوّل.

8- اكتب برنامجاً يعرف صنف مستطيل Rectangle ، يتضمن خاصتين هما الطول length والعرض width من نوع double . كما يتضمن تابعان الأول لإنشاء مستطيل createRectangle() وله بارامبتران هما طول وعرض المستطيل والثاني لحساب مساحة المستطيل calculateArea() بدون بارامبترات ولكنها ترجع بعد استدعائها مساحة المستطيل.

9- اكتب برنامجاً يعرف الصنف ساعة رقمية DigitalClock الذي يتضمن ثلاث خصائص هي الساعات hours والدقائق minutes والثواني seconds من نوع عدد صحيح integer . كما يتضمن هذا الصنف تابعين : الأول لضبط الوقت setTime() وله ثلاثة بارامبترات هي الساعة ويمكن أن تأخذ القيم من 1 إلى 12 والدقيقة وتأخذ القيم من 0 إلى 59 أما البارامبتر الثالث فهو

لإظهار حالتين هما قبل الظهر AM وبعد الظهر PM. أما التابع الثاني فهو لإظهار الوقت showTime() حيث يظهر الوقت. ثم يقوم البرنامج بإنشاء ثلاثة كائنات من هذا الصنف ويستخدم توابع الصنف للوصول إلى هذه الكائنات وتغيير خصائصها وأخيراً إزالتها.

مسائل برمجية محلولة

1- ما هي قيم المتحولات x,s,w بعد تنفيذ البرنامج التالي، اشرح باختصار آلية التنفيذ:

```
# include < iostream.h >
main()
{
    int x = 1, s,w;
    s = w = 0;
    do
    {
      if (x > = 4)
          s + = x/2;
      else
          w+ = x;
      if (x > = 4)
          x+ = 1;
      else
          x+ = 1;
    } while (x < = 8);
    cout << x << " , " << s << " , " << w << endl;
    return 0;
}
```

يبدأ البرنامج بقيمة لـ x = 1 وقيمة لكل من w , s تساوي الصفر وطالما أن (x<=8) يكرر اختبار قيمة x الجديدة.

فإذا كانت (x > = 4) ينفذ البرنامج حساب s من العلامة s + = x/2 وإلاّ ينفذ حساب w من العلامة w+ = x.

ثم يعود ويختبر x مرة أخرى فإذا كانت (x > = 4) ينفذ حساب x من العلاقة x+=1 وإلا يحسب قيمة x الجديدة من العلامة x+ = 1 أيضاً. وبتنفيذ البرنامج تصبح قيم x = 9 و s = 14 و w = 6.

2- ما هو خرج البرنامج التالي، اشرح خطوات تنفيذ البرنامج باختصار :

```
# include <iostream.h>
main()
{
    for (int i=0; i<8, i++)
      if (i%2 = = 0)
           cout << i+1 << " , ";
      else if (i%3 = = 0)
             cout << i*i<< " , ";
        else if (i%5 = = 0)
               cout << 2*i-1 << " , ";
             else
                 cout << i << ",";
    return 0;
}
```

يقوم البرنامج باختبار قيم i التي تتغير من 0 وإلى 7 بتزايد 1 حيث يختبر كل قيمة جديدة لـ i ، فإذا كانت من مضاعفات العدد 2 يطبع القيمة i+1،

وإذا كانت من مضاعفات العدد 3 يطبع القيمة i*i

وإذا كانت من مضاعفات العدد 5 يطبع القيمة 2*i-1 وإلا يطبع قيمة i فقط.

يتوقف التنفيذ عندما تصبح قيمة i تساوي 8.

وخرج البرنامج هو :

```
1 , 1 , 3 , 9 , 5 , 9 , 7 , 7
```

حيث من أجل i=0 فإن الاختبار (i%2= =0) محقق لذلك يطبع قيمة (i+1=1)

ومن أجل i=1 فإن الاختبارات الثلاث غير محققة لذلك يطبع قيمة (i=1)

ومن أجل i=2 فإن الاختبار (i%2= = 0) محقق لذلك يطبع القيمة (i+1=3)

وهكذا حتى تصبح i=7 عندها الاختبارات كلها غير محققة لذلك يطبع القيمة (i=7) وعندما i=8 يتوقف التنفيذ.

3- ما هو ناتج تنفيذ كل مما يلي، اشرح باختصار عمل كل منهما :

a)

```
# include<iostream.h>
main()
{
    int f0,f1,f2;
    f0=f1=f2=1;
    for(int i=0; i<10; i++)
      { f0=f1;
        f1=f2;
        f2=f0+f1;
        cout<<f2<<"    ";}
    return 0;
}
```

b)

```
# include <iostream.h>
main()
{
    int n=5,x;
    x=++n*--n;
    cout<<"n="<<n<<",x="<<x<<endl;
    cout<<++n<<" ,  " <<++n+3<<" ,  "
         <<++n-5<<endl;
  return 0;
}
```

a- يصرح البرنامج عن ثلاثة متغيرات صحيحة ويعين لها قيماً ابتدائية مساوية للواحد. ومن أجل قيم المتغير i من 0 إلى 9 يتكرر تعيين قيم جديدة لكل من f0,f1,f2 وتطبع قيم f2 الناتجة في سطر واحد. وهي كالتالي:

```
2   3   5   8   13   21   34   89  144
```

b- يصرح البرنامج عن متغير صحيح n ويعين له القيمة 5، وآخر x ويعين له ناتج حساب صيغة تستخدم معاملات حسابية وتزايد أو تناقص سابق أو لاحق. والخرج هو مايلي:

```
n = 5  , x = 25
6  ,  10  ,  3
```

4- اكتب برنامجاً لإيجاد كل الأعداد المؤلفة من ثلاثة أرقام والتي تخضع للخاصية التالية :

العدد = مجموع مكعبات أرقامه

مثال :

$$\begin{aligned} 371 &= 3^3 + 7^3 + 1^3 \\ &= 27 + 343 + 1 = 371 \end{aligned}$$

```
# include <iostream.h>
main()
{
    int sum, a, c;
    for (int i=100; i<1000; i++)
      {
        c=i;
        sum=0;
        while (c! = 0)
          {
            a = c% 10;
            c = c/ 10;
            sum = sum + a * a * a;
            if (( c = = 0) && (sum = = i))
              cout << " i = " << i << "\t";
          }
      }
    return 0;
}
```

وخرج البرنامج هي الأعداد المكونة من ثلاثة أرقام وتحقق الخاصية السابقة وهي:

i = 153 i = 370 i = 371 i = 407

5- إذا أعطي عدداً صحيحاً n1 فإنه يمكن تشكيل مصفوفة الأعداد n1,n2,n3,... باتباع القاعدة التالية :

$n_{i+1} =$ مجموع مربعات أرقام n_i

وأثناء تشكيل مثل هذه المصفوفة سنصادف إحدى الحالتين التاليتين :

1- تشكل الرقم 1 وبالتالي كل الأعداد المشكلة بعدها تساوي الواحد.

2- الحصول على العدد 4 والأعداد المشكلة بعدها هي :

4 , 20 , 42 , 145 , 89 , 58 , 37 , 16

وتستمر المصفوفة بتكرار هذه الأعداد ولا يتشكل العدد 1 أبداً.

اكتب برنامجاً يقبل كدخل عدداً صحيحاً موجباً ثم يقوم بتشكيل مصفوفة الأعداد حسب القاعدة السابقة إلى أن يصادف العدد 1 أو العدد 4 .

```
# include <iostream.h>
main()
{
      int sum = 0, a , b , num, i=0, s[10];
      cout << " enter a positive integer number:";
start: cin >> num;
      if (num < 0)
      goto start;
      a = num;
      s[i] = num;
stat:  do
      {
        b = a% 10;
        a = a/10;
```

```
          sum = sum + b*b;
        } while (a ! = 0);
          i++;
          a = s[i] = sum;
          sum = 0;
          if ((s[i] = = 1) || (s[i] = = 4))
          goto end;
        else
          goto stat;
end:  cout << "display array s : " << endl;
        for (int j = 0; j < = i; j++)
          cout << s[j] << " , ";
        cout << endl;
    return 0;
}
```

والخرج هو :

```
enter a positive integer number: 8⏎
display array s:
8 , 64 , 52 , 29 , 85 , 89 , 145 , 42 , 20 , 4
```

6- اكتب برنامجاً للبحث عن ثلاثة أعداد صحيحة x,y,z ضمن المجال (1-50) بحيث تحقق الخاصية :

$$z^2 = x^2 + y^2$$

ثم طباعة تلك الأعداد، تأكد من عدم طباعة مجموعة الأعداد التي تعتبر نسخة من مجموعة أخرى ، أي (3,4,5) و (4,3,5) و (5,3,4) ...

```
# include <iostream.h>
main()
{
        int i,j,k;
        for (i=1; i<=50; i++)
          for (j=i+1; j<=50; j++)
```

```
              for (k=j+1; j<=50; k++)
          if((i*i= =j*j+k*k) || (j*j= =i*i+k*k) || (k+k= =i*i+j*j))
            cout << i << " , " << j << " , " << k << endl;
     return 0;
}
```

وخرج البرنامج هو :

```
3 , 4 , 5
5 , 12 , 13
6 , 8 , 10
7 , 24 , 25
8 , 15 , 17
9 , 12 , 15
9 , 40 , 41
10 , 24 , 26
12 , 16 , 20
12 , 35 , 37
14 , 48 , 50
15 , 20 , 25
15 , 36 , 39
16 , 30 , 34
18 , 24 , 30
20 , 21 , 29
21 , 28 , 35
24 , 32 , 40
27 , 36 , 45
30 , 40 , 50
```

7- اكتب برنامجاً دخله عدد صحيح بالترميز عشري وخرجه مكافئ للعدد المدخل ولكن بالترميز الثنائي.

```
# include <iostream.h>
main()
{
        int num10, b, c, i=0;
        int num2[10];
        cout << " enter a decimal integer number:";
        cin >> num10;
        int num = num10;
start: b = num10/2;
        c = num10%2;
        num2[i] = c;
        i++;
        if (b = = 0)
          goto end;
        else
          {
            num10 = b;
            goto start;
          }
end: cout << "the decimal number" << num << " is equal to";
        for (int j=i-1; j>=0; j--)
          cout << num2[i];
        cout << "in binary " << endl;
    return 0;
}
```

والخرج هو :

```
enter a decimal integer number: 13↵
the decimal numbeer 13 is equal to 1101 in binary
```

8- اكتب برنامجاً يقوم بإدخال قيمة لمتغير صحيح B . ثم يقوم باستخدام تعليمة switch لحساب ما يلي :

أ – إذا كانت قيمة B = 1 يسمح بإدخال قيمة نصف قطر دائرة x ، ويحسب ويظهر مساحتها عن طريق استدعاء تابع فرعي أول.

ب – إذا كانت قيمة B = 2 يسمح بإدخال قيمة طول القاعدة x ، وقيمة الارتفاع H لمثلث، ثم يحسب ويظهر مساحة المثلث المشكل منهما عن طريق استدعاء تابع فرعي ثاني.

ج – إذا كانت قيمة B = 3 يسمح بإدخال قيمة طول ضلع مربع x ، ثم يحسب ويظهر مساحة المربع عن طريق استدعاء تابع فرعي ثالث.

د – يتم إنهاء البرنامج إذا كانت قيمة B لاتساوي أي من القيم السابقة.

```
# include <iostream.h>
# include <math.h>
void circle();
void triangle();
void square();
main()
{
      int b;
      cout << " enter value for b:";
      cin >> b;
      switch (b)
        {
          case1: circle(); break;
          case2: triangle(); break;
          case3: square (); break;
        }
      return 0;
}
void circle()
      {
        float g, p=3.14;
        int x;
        cout << " enter radius x:";
        cin >> x;
```

```
           g = p*pow(x,2);
           cout << "Area circle = " << g << endl;
        }
void triangle()
        {
           float x,h,z;
           cout << " enter base and hight x,h << endl;
           cin>>x>>h;
           z = 0.5 * x * h;
           cout << " Area triangle = " << z << endl;
        }
void square()
{
        float x,y;
        cout << " enter side x:";
        cin >> x;
        y = pow (x,2);
        cout << " Area square = " << y << endl;
}
```

والخرج هو :

```
enter value for b:1↵
enter radius x : 3↵
Area circle = 28.26
```

9- اكتب برنامجاً يقارن عناصر مصفوفتين b , a كل منهما ذات بعدين ولهما العدد من العناصر نفسه . ويكون مصفوفة ثالثة c كل عنصر فيها يكون 1- أو صفر أو 1 على حسب ما إذا كان العنصر المقابل في المصفوفة a أكبر من أو يساوي أو أقل من العنصر المقابل في المصفوفة b.

```
# include < iostream.h >
# include <iomanip.h>
main ( )
{
```

```
int a[2] [3] , b[2] [3], c[2] [3];
cout << "enter array a and display :";
for (int i = 0;i < 2; i++)
    {
        for (int j = 0; j < 3; j++)
             {
                  cin >> a[i] [j];
                  cout << a[i] [j] << ",";
             }
        cout << endl;
    }
cout << "enter array b and display ":;
for ( i = 0; i < 2; i++)
    {
        for (j = 0; j < 3; j++)
             {
                  cin >> b[i] [j];
                  cout << b[i] [j] << ",";
             }
        cout << endl;
    }
cout << "compare a and b and display c:" << endl;
for ( i = 0; i < 2; i++)
    {
        for (j = 0; j < 4; j++)
             {
                  if (a[i] [j] > b[i] [j])
                       c[i] [j] = 1;
                  else  if ( a[i] [j] < b[i] [j]
                       c[i] [j] = -1;
                  else
                       c[i] [j] = 0;
                  cout << setw(5) << c[i] [j];
```

```
                }
            cout << endl;
        }
    return 0;
}
```

والخرج هو :

```
enter array a and display: 22   44   66   50   50   50↵
22   44   66
50   50   50
enter array b and display: 33   44   55   50   50   80↵
33   44   55
50   50   80
Compare a and b and display c:
  -1   0   1
   0   0  -1
```

10- اكتب برنامجاً يسمح بإدخال قيمٍ صحيحة لعناصر مصفوفة ذات بعدين m و n ثم يخرجها بعد التعديل على النحو التالي :

يقوم بطرح أصغر عنصر في كل سطر من جميع عناصر ذلك السطر وكذلك يقوم بطرح أصغر عنصر في كل عمود من عناصر ذلك العمود.

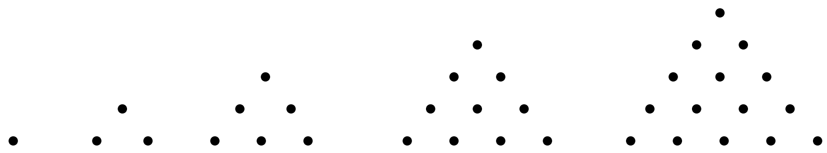
```
# include < iostream.h >
main ( )
{
    int a[5] [5] , minr[5], minc[5], n, m, i, j;
    cout << " enter n and m values:";
    cin >> n >>m;
    cout << " enter array a and display : " << endl;
       for ( i = 0; i < n; i++)
           {
               for ( j = 0; j < m; j++)
                   {
                       cin >> a[i] [j];
```

```
                    cout << a[i] [j] << "      ";
                }
            cout << endl;
        }
    cout << endl;
    for ( i = 0; i < n; i++)
        {
            minr[i] = a[i] [0];
            for (j = 1; j < m; j++)
                {
                    if (a[i] [j] < minr[i])
                        minr[i] = a[i] [j];
                }
            for (j = 0; j < m; j++)
                a[i] [j] = a[i] [j]-minr[i];
        }
    for ( j = 0; j < m; j++)
        {
            minc [j] = a[0] [j];
            for (i = 1; I < n; i++)
                {
                    if (a a[i] [j] < minc [j])
                        minc[j] = a[i] [j];
                }
            for (i = 0; i < n; i++)
                a[i] [j] = a[i] [j]-minc[j];
        }
    cout << " the modified array a is": << endl;
    for (i = 0; i < n; i++)
        {
            for (j = 0; j < m; j++)
                cout << a[i] [j] << "      ";
            cout << endl;
        }
    return 0;
}
```

والخرج هو :

```
enter n and m values : 3  4
enter array a and display:
1  3  2  4  5  2  6  9  4  6  3  4
1  3  2  4
5  2  6  9
4  6  3  4
the modified array a is :
0  2  1  2
3  0  4  6
1  3  0  0
```

11- العدد المثلثي هو الذي يمكن أن يمثل على شكل مثلث متساوي الأضلاع، كما يلي:



$T_1 = 1$ $T_2 = 3$ $T_3 = 6$ $T_4 = 10$ $T_5 = 15$

المطلوب كتابة برنامج يدخل عدداً صحيحاً n ، ثم يستدعي تابعاً فرعياً triamgle ليختبر إذا كان هذا العدد مثلثي عندها يرجع الرقم 1 ، وإلاّ فإنه يرجع الرقم 0. ومن ثم يخرج البرنامج إحدى الرسالتين التاليتين:

n is triangle	إذا كان العدد مثلثياً.
n is not triangle	إذا لم يكن العدد مثلثياً.

```
# include <iostream.h>
int triangle (int);
main()
{
    int n;
    do
    {
       cout << " enter integer value:";
       cin >> n;
       if (triangle (n))
           cout << n << " is triangle " << endl;
       else
           cout << n << " is not triangle " << endl;
    } while ( n > 0 );
  return 0;
}
int triangle (int n)
{
    int i = 0, sum = 0;
    while (sum < n)
       sum + = i++;
    if (sum = = n)
       return 1;
    else
       return 0;
}
```

والخرج هو :

```
enter integer value: 15↵
15   is triangle
enter integer value: 12↵
12  is not triangle
Enter integer value: -1↵
-1 is not triangle
```

12- اكتب برنامجاً يطبع جميع الأعداد الصحيحة والتامة والواقعة ضمن المجال (1000-1) ويستخدم تابعاً فرعياً اسمه perfect() يتحقق من كون العدد الممرر إليه عدداً صحيحاً وتاماً.

ملاحظة : يكون العدد الصحيح عدداً تاماً، إذا كان مجموع عوامله بما فيها الواحد (ماعدا العدد نفسه) يساوي العدد نفسه. مثلاً العدد 6 هو عدد تام حيث: 1+2+3=6.

```
# include <iostream.h>
int perfect (int);
main()
{
    int i,n;
    for (i=1; i<=1000; i++)
       {
          n=perfect (i);
          if(i= = n)
               cout << i << " is perfect number. " << endl;
       }
    return 0;
}
int perfect (int k)
{
    int i, sum = 0;
    for (i=1; i<=k/2; i++)
       {
          if(k%i = = 0)
          sum = sum +i;
       }
    if (k = = sum)
       return k;
    else
       return 0;
}
```

والخرج هو:

```
6      is perfect number.
28     is perfect number.
496    is perfect number.
```

13- اكتب برنامجاً يدخل قيم العناصر مصفوفة أحادية ويطبعها كما أدخلت. ثم يستدعي التابع reverse لعكس عناصرها من خلال استخدام مصفوفة المؤشرات التي تشير إلى عناصر المصفوفة المدخلة.

```
# include < iostream.h >
const int size = 20;
void enterarray (int*, int&);
void printarray (int*, int);
void reverse (int* [ ], int*, int);
main()
{
     int a [size];
     int * index [size];
     int m;
     cout << " enter array a:";
     enterarray (a,m);
     cout << " array a is :";
     printarray (a,m);
     for (int i=0; i<m; i++)
        index [i] = & a [i];    // assign array pointer to array a elements
     reverse (index , a,m);
     cout << " array a after reverse reversing its elements is:" << endl;
     for ( i = 0; i < m; i++)
        cout << * index [i] << "   ";
     cout << endl;
   return 0;
}
void enterarray (int * b, int& k)
{
     k = 0;
     for (int * p = b; p < b + size; p++)
        {
            cin >> * p;
            if (*p = = 0)
            break;
            k++;
        }
}
void printarray (int*b, int k)
{
    for (int*p = b; p < b + k; p++)
            cout << * p << "   ";
    cout << endl;
}
void reverse (int* pointers [ ], int * b, int k)
{
    for (int i = 0; i < k/2; i++)
        {
            int * temp = pointers [i];
            pointers [i] = pointers [k-i-1];
            pointers [k-i-1] = temp;
        }
}
```

والخرج هو :

```
enter array a : 1   2   3   4   5   6   7   0↵
array a is : 1   2   3   4   5   6   7
array a after reversing its elements is:
7   6   5   4   3   2   1
```

14- اكتب برنامجاً يقوم بإدخال عناصر مصفوفة ثم يطبعها كما أدخلت ومن ثم يدور عناصرها نحو اليمين بمقدار f (أو نحو اليسار بمقدار f–) . استدع التوابع التالية بالشكل المناسب واعتمد على المؤشرات في الحل.

1) التابع enterarray لإدخال عناصر مصفوفة a حدها الأعظمي 20 عنصر ويتم التحكم بعدد العناصر المدخلة بإدخال القيمة صفر.

2) التابع printarray لطباعة المصفوفة a.

3) التابع rotate يقوم بدوران عناصر المصفوفة a بمقدار f من الأماكن ناحية اليمين (أو f من الأماكن ناحية اليسار إذا كانت f سالبة).

مثال على ذلك : إذا كانت لدينا المصفوفة a هي :

10 20 30 40 50 60

فإذا كان المطلوب الإزاحة بمقدار f = 3 عندئذ تصبح المصفوفة كما يلي:

40 50 60 10 20 30

أما لو كان المطلوب الإزاحة بمقدار f = -2 عندئذ تصبح المصفوفة كما يلي:

30 40 50 60 10 20

```
# include <iostream.h>
const int size = 20;
void enterarray (int*, int&);
void printarray (int*, int);
void rotate (int*, int, int);
main()
{
    int b[size];
    int m,f;
    char ch;
    while (1)
    {
      cout << " enter array b:";
      enterarray (b, m);
      cout << " array b is:";
      printarray (b,m);
      cout << " enter value for f to rotate b with";
      cin >> f;
      if ( f < m )
       {
         rotate (b, m, f);
```

```
              cout << " array b after rotate is:";
              printarray (b,m);
           }
         else
           cout << " error value for f.";
      cout << " press y to continue and another key to finish:";
      cin >> ch;
      if (ch ! = 'y')
          break;
      }
  return 0;
}
void enterarray (int * a, int & k)
      {
          k = 0;
          for (int * p = a; p < a + size; p++)
               {
                    cin >> * p;
                    if (*p = = 0)
                       break;
                    k++;
               }
}
void printarray (int * a, int k)
      {
          for (int * p = a; p < a + k , p++)
               cou << * p << "    ";
          cout << endl;
      }
void rotate (int * a, int k, int n)
      {
          int i, d[size];
          if (n < 0)
```

```
    n = k + n;
    for (int * p = a, * q = d; p < a + k; p++, q++)
        * q = * p;
    for ( p = a, q = d, i = 0; p < a + k; p++, i++)
        if ( i < n)
            * p = q [ k-n+i];
        else
        *p = q [i-n];
    }
```

والخرج هو :

```
enter array b : 1   2   3   4   5   6   0↵
array b is : 1   2   3   4   5   6
enter value for f to rotate b : 2↵
array b after rotate is : 5   6   1   2   3   4
press y to continue and another key to finish : y↵
enter array b : 10   20   30   40   50   70   90   0↵
array b is : 10   20   30   40   50   70   90
enter value for f to rotate b : -3↵
array b after rotate is :  40   50   70   90   10   20   30
press y to continue and another key to finish : n↵
```

15- اكتب برنامجاً يقوم بإدخال n عنصر للمصفوفة a ثم يطبعها كما أدخلت. ومن ثم يستدعي التابع append الذي يضيف m عنصر من المصفوفة b إلى نهاية المصفوفة المدخلة. استدع التوابع printarray, enterarray المستخدمين في المثال (13) والتابع append بالشكل المناسب لتحقيق المطلوب.

ملاحظة : التابع append يفترض أن المصفوفة a تتسع لعدد m + n عنصر على الأقل.

```
#include<iostream.h>
const int size = 50;
void enterarray (int*, int&);
void printarray (int*, int);
void append (int*, int&, int*, int);
```

```
main()
{
     int a[size], b[size];
     int n , m;
     cout << "enter array a:";
     enterarray (a,n);
     cout << "enter array b:";
     enterarray (b,m);
     cout << "array a is:";
     printarray (a,n);
     cout << " array b is:";
     printarray (b,n);
     append (a,n,b,m);
     cout << " array a after append is:" << endl;
     printarray (a,n);
   return 0;
}
void enterarray (int* a, int&n)
     {
        n = 0;
        for (int*p = a; p < a + size; p++)
            {
                cin >> * p;
                if (*p = = 0)
                break;
                n++;
            }
     }
void printarray (int*a, int n)
     {
        for (int * p = a; p < a +n; p++)
            cout << * p << "    ";
        cout << endl;
```

```
    }
void append (int*a, int&n, int*b, int m)
    {
       for (int*p = a + n, *q = b; q < b + m; p++, q++)
           {
               *p = * q;
               n ++;
           }
    }
```

والخرج هو:

```
enter array a : 1   2   3   0↵
enter array b : 4   5   6   7   0↵
array a is : 1   2   3
array b is : 4   5   6   7
array a after append is :
1   2   3   4   5   6   7
```

16- اكتب برنامجاً يحول المصفوفة A[5][5] وعناصرها القيم من 1-25 إلى المصفوفة B المبينة في الشكل التالي:

$$\begin{matrix} 1 & 2 & 3 & 4 & 5 \\ 16 & 17 & 18 & 19 & 6 \\ 15 & 24 & 25 & 20 & 7 \\ 14 & 23 & 22 & 21 & 8 \\ 13 & 12 & 11 & 10 & 9 \end{matrix} \quad \Leftarrow \quad \begin{matrix} 1 & 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 & 10 \\ 11 & 12 & 13 & 14 & 15 \\ 16 & 17 & 18 & 19 & 20 \\ 21 & 22 & 23 & 24 & 25 \end{matrix}$$

المصفوفة B **المصفوفة A**

```
# include <iostream.h>
# include <iomanip.h>
main()
{
      int a[5][5]={1,2,3,4,5},{6,7,8,9,10},{11,12,13,14,15},
                   {16,17,18,19,20},{21,22,23,24,25}};
      int b[5][5];
      cout << "\t display input array:"<<"\n"<<endl<<"\t";
```

```
for (int i=0; i<5; i++)
  {
    for (int j=0; j<5; j++)
      cout << setw(6) << a [i][j];
    cout << endl << "\t";
  }
  cout << endl;
  cout << "\t display output array:"<<"\n"<<endl<<"\t";
  for (i=0; i<5; i++)
    for (j=0; j<5; j++)
      {
          b[i][j] = a[i][j];
          if(i = = 0)
            continue;
          if ((i = = 1) && (j ! = 4))
            b[i][j] = b [i-1][j]+15;
          if((i!= 0)&&(j = = 4))
            b[i][j] = b[i-1][j]+1;
          if(i= =1)
            continue;
          if((i>=2)&&(j = = 0))
            b[i][j]=b[i-1][j]-1;
          if ((i = = 2)&&((j = = 1) || (j = = 2)))
            b[i][j] = b[i-1][j]+7;
          if (((i = = 2) || (i = = 3))&&(j = = 3))
            b[i][j] = b[i-1][j]+1;
          if(i = = 2)
            continue;
          if((i= = 3)&&(j = = 1))
            b[i][j] = b[i-1][j]-1;
          if((i = = 3)&&(j= =2))
            b[i][j]=b[i][j-1]-1;
          if (i = = 3)
```

```
                continue;
              if((i = = 4)&&(j ! = 0))
                b[i][j] = b[i][j-1]-1;
          }
              for (i = 0; i<5; i++)
                {
                  for (j=0; j<5; j++)
                    cout << setw(6)<<b[i][j];
                  cout << endl << "\t";
                }
     return 0;
}
```

وخرج البرنامج هو:

```
display input array
        1      2      3      4      5
        6      7      8      9     10
       11     12     13     14     15
       16     17     18     19     20
       21     22     23     24     25
display output array:
        1      2      3      4      5
       16     17     18     19      6
       15     24     25     20      7
       14     23     22     21      8
       13     12     11     10      9
```

17- اكتب برنامجاً يدخل قيماً لعناصر المصفوفة الثنائية a[5][5] ويعيد تنظيمها بحيث تصبح أسطرها وأعمدتها الحاوية على صفر صفراً.

مثال:

$$\begin{array}{ccccc} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 6 & 0 & 0 & 3 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 5 & 0 & 0 & 9 \end{array} \quad \Leftarrow \quad \begin{array}{ccccc} 1 & 2 & 7 & 0 & 2 \\ 0 & 4 & 2 & 3 & 1 \\ 3 & 6 & 8 & 9 & 3 \\ 7 & 3 & 0 & 6 & 5 \\ 1 & 5 & 4 & 7 & 9 \end{array}$$

المصفوفة a المدخلة **المصفوفة a المعدلة**

طريقة الحل:

نستعين بمصفوفة أحادية m[5] ونعين لعناصرها قيمة ما لاتساوي الصفر، ثم نبحث في المصفوفة a عن الأسطر التي تحوي على عنصر أو أكثر مساوي للصفر عندها نصفر عناصر تلك الأسطر. ولتحديد رقم العمود الذي يقع فيه العنصر المساوي للصفر في المصفوفة a (لتصغير قيم عناصره كما هو مطلوب) نستعين بالمصفوفة m ونعين لعنصر m الذي دليله عمود العنصر في a والذي قيمته صفراً ، نعين له قيمة تساوي قيمة دليل العنصر المساوي للصفر في a بتنفيذ التعليمة :

```
if(a[i][j] = = 0)
m[j] = j;
```

بعد ذلك نبحث عن عناصر المصفوفة m التي لاتساوي قيمها القيمة الأولية 7 ونصفر عناصر أعمدة المصفوفة a والتي دليلها مساوي لدليل عناصر المصفوفة m والتي لم تعد مساوية لـ 7.

```
# include <iostream.h>
# include <iomanip.h>
main()
{
      int a[5][5]={{1,2,7,0,2},{0,4,2,3,1},{3,6,8,9,3},{7,3,0,6,5},
            {1,5,4,7,9}};
      int m[5]={7,7,7,7,7};
      cout << " display array a : " << endl;
      for (int i=0; i<5; i++)
```

```
  {
    for (int j=0; j<5; j++)
    cout << setw(4) << a[i][j];
    cout << endl;
  }
cout << " display array m : " << endl;
for (i=0; i<5; i++)
  cout << setw(5) << m[i];
cout << endl;
for ( i=0; i<5; i++)     // set to zero all row's elements in array a
  {                          // if it has a zero element in it
    for (int j=0; j<5; j++)
    {
      if(a[i][j] = = 0)
      {
          m[j] = j;
          for (int k=0; k<5; k++)
            a[i][k] = 0;
          break;
      }
  }
}
cout << endl;
cout << " display array a with zero row's : " << endl;
for (i=0; i<5; i++)
  {
    for (int j=0; j<5; j++)
      cout << setw(4) << a [i][j];
    cout << endl;
  }
cout << " see the changed elements in m:"<< endl;
for (i=0; i<5; i++)
  cout << setw(4) << m[i];
```

```
        cout << endl;
        for (j=0; j<5; j++)       // set array a coloum to zero
        {
          if (m[j] ! = 7)
          {
            for (int i=0; i<5; i++)
              a[i][m[j]] = 0;
          }
        }
        cout << " display output array a:"<< endl;
        for (i=0; i<5; i++)
          {
            for (j=0; j<5; j++)
              cout << setw(4) << a[i][j];
            cout << endl;
          }
        return 0;
}
```

والخرج هو :

```
display array a:
1   2   7   0   2
0   4   2   3   1
3   6   8   9   3
7   3   0   6   5
1   5   4   7   9
display array m:
    7     7     7     7     7
display array a with zero rows:
    0     0     0     0     0
    0     0     0     0     0
    3     6     8     9     3
    0     0     0     0     0
    1     5     4     7     9
```

```
see the changed elements in m:
    0      7      2      3      7
display output array a:
    0      0      0      0      0
    0      0      0      0      0
    0      6      0      0      3
    0      0      0      0      0
    0      5      0      0      9
```

18- اكتب برنامجاً يطبع المثلث الموضح أدناه. استخدم تعليمات الطباعة التي تكرر طباعة الرمز نجمة '*' أو رمز الفراغ ' ' . كذلك استخدم أقل ما يمكن من تعليمات الطباعة وأكثر ما يمكن من التكرار باستخدام الحلقات المتداخلة :

```
        *
      * * *
    * * * * *
  * * * * * * *
* * * * * * * * *
```

```
# include < iostream.h >
main ()
{
    for (int i=0; i<5; i++)
       {
          for (int j=1; j<10; j++)
              {
                 if (j = = 10/2-i)
                     for (int k=10/2-i; k<=10/2+i; k++)
                             cout << '*';
                 else
                     cout << '  ';
              }
          cout << endl;
       }
    return 0;
}
```

19- عمم المثال السابق لطباعة المعين الموضح أدناه. استخدم تعليمات الطباعة التي تكرر طباعة الرمز نجمة '*' أو رمز الفراغ ' ' . استخدام أقل ما يمكن من تعليمات الطباعة وأكثر ما يمكن من التكرار باستخدام الحلقات المتداخلة .

```
        *
      * * *
    * * * * *
  * * * * * * *
* * * * * * * * *
  * * * * * * *
    * * * * *
      * * *
        *
```

```
# include < iostream.h >
main ()
{
    for (int i=0; i<9; i++)
       {
          for (int j=1; j<10; j++)
             {
                if ((i<10/2) && (j = = (10/2-i)))
                    {
                       for (int k=(10/2-i); k<=(10/2+i); k++)
                           cout << '*';
                       cout << '  ';
                       continue;
                    }
                if (( i>=10/2) && ( j = = i-3))
                    for (int k=i-3; k<=10-(i-3); k++)
                       cout << '*';
                else
                    cout << '  ';
             }
          cout << endl;
       }
    return 0;
}
```

ويكون الخرج معيناً من النجوم طول كل من قطرية تسعة نجوم.

20- أعد كتابة المثال (8) من تمارين آخرالفصل السابع، ولكن باستخدام الحلقات في التكرار

```
# include < iostream.h >
main ( )
{
     int num, n[100], i, k = 0;
     cout << " enter an integer positive number:";
start: cin >> num;
     if (num < = 0)
     goto start;
     n[k] = num;
     do
        {
           if (n[k]%2 = = 0)
                n[k+1] = n[k]/2;
           else
                n[k+1] = n[k] * 3+1;
           k+ = 1;
        }  while (n[k]! = 1);
     cout << "number of array elements k = " << ++k<<endl;
     for (i = 0; i < k; i++)
        cout << n[i] << ",";
     cout << endl;
     return 0;
}
```

والخرج هو :

```
enter an integer positive number:3↵
number of array element k =8
the array elements are:
3 , 10 , 5 , 16 , 8 , 4 , 2 , 1,
```

21- استخدم مصفوفة ثنائية البعد في حل التمرين التالي:

شركة تبيع خمس مواد مختلفة، وتستخدم لذلك أربعة عمال. إذا كان كل عامل يقدم في نهاية كل يوم لصاحب الشركة ما بين (0-5) أوراق، وكل ورقة مدون عليها المعلومات التالية عن كل مادة باعها:

1- رقم البائع.

2- رقم المادة المباعة.

3- إجمالي قيمة ما باعه هذا البائع من المادة المذكورة.

بفرض أن المعلومات متوفرة عن كل الأوراق المقدمة لصاحب الشركة خلال شهر (30 يوم) .

اكتب برنامجاً يقوم بقراءة كل المعلومات المتوفرة خلال شهر واحد ويخزن هذه القيم في مصفوفة ثنائية اسمها sales، ومن ثم يطبع تلك القيم في مصفوفة بحيث يمثل كل سطر إحدى المواد المباعة، وكل عمود قيمة ماباعه كل بائع من كل مادة. ثم يقوم بجمع قيم كل سطر للحصول على مجموع أثمان ما بيع من كل مادة خلال شهر، وكذلك يقوم بجمع قيم كل عمود للحصول على مجموع أثمان ما باعه كل بائع في الشهر. اطبع مجموع كل سطر على يمين السطر، ومجموع كل عمود في أسفله.

```
# include < iostream.h >
# include < iomanip.h >
main ( )
{int Sales[5][4], Salesr[5];
 int Salec [4], Salesd[30];
 for(int i=0;i<5;i++)
   { for(int j=0;j<4;j++)
       { Sales[i][j]=0;
          for(int k=0;k<30;k++)
            { cin>>Salesd[k];
                Sales[i][j]=Sales[i][j]+salesd[k];
```

```
             }
          cout<<Sales[i][j]<<"    ";
       }
     cout<<endl;
   }
cout<<endl;
for(i=0;i<5;i++)
 { Salesr[i]=0;
    for(int j=0;j<4;j++)
      { cout<<setw(10)<<Sales[i][j];
         Salesr[i]=Salesr[i]+Sales[i][j];
       }
   cout<<setw(10)<<Salesr[i];
   cout<<endl;
  }
cout<<endl;
for(int j=0;j<4;j++)
 { Salesc[j]=0;
    for(int i=0;i<5;i++)
      { Salesc[j]=Salesc[j]+Sales[i][j];
      cout<<setw(10)<<Salesc[j];
      }
 }
return 0;
}
```

الملاحق

الملحق A

شفرات ASCII

كل حرف يخزن في صورة شفرات ASCII ، وهذه الشفرات عبارة عن أرقام صحيحة من صفر حتى 127. أول 32 حرف هي عبارة عن أحرف غير مكتوبة، لذلك فإن رموزهم في العمود الأول تكون مصحوبة إما بالأحرف "Ctrl" أو بالحرف '\' أو الشرطة العكسية. في الحالة الأولى نضغط المفتاح كونترول ctrl مع المفتاح الثاني للحصول على الحرف غير المكتوب. فمثلاً حرف نهاية الملف (end-of-file) والذي شفرته هي 4 يتم إدخاله بالضغط على الزر ctrl وفي نفس الوقت نضغط الزر D .

في الحالة الثانية نكتب الحرف '\' متبوعاً بالحرف المطلوب، فمثلاً في لغة C++ الشفرة "\" هي شفرة السطر الجديد newline وشفرتها هي 10.

Character	Description	Decimal	Octal	Hex	Binary
Ctrl-@	Null, end of string	0	0	0x0	0000 0000
Ctrl-A	Start of heading	1	01	0x1	0000 0001
Ctrl-B	Start of text	2	02	0x2	0000 0010
Ctrl-C	End of text	3	03	0x3	0000 0011
Ctrl-D	End of transmission, end of file	4	04	0x4	0000 0100
Ctrl-E	Enquiry	5	05	0x5	0000 0101
Ctrl-F	Acknowledge	6	06	0x6	0000 0110
\ a	Bell, alert, System beep	7	07	0x7	0000 0111
\ b	Backspace	8	010	0x8	0000 1000
\ t	Horizontal tab	9	011	0x9	0000 1001
\ n	Line feed, new line	10	012	0xa	0000 1010
\ v	Vertical tab	11	013	0xb	0000 1011
\ f	Form feed, new page	12	014	0xc	0000 1100
\ r	Carriage return	13	015	0xd	0000 1100
Ctrl-N	Shift out	14	016	0xe	0000 1110
Ctrl-O	Shift in	15	017	0xf	0000 1111
Ctrl-O	Data link escape	16	020	0x10	0000 0000

Ctrl-Q	**Device control 1,resume scroll**	**17**	**021**	**0x11**	**0000 0001**

1- ASCII هي اختصار للعبارة (الشفرات الأمريكية القياسية لتبادل المعلومات) أو Americal Standard Code for Information Interchange

تابع شفرات ASCII

Character	Description	Decimal	Octal	Hex	Binary
Ctrl - R	Devic control 2	18	022	0x12	0001 0010
Ctrl - S	Device control 3, stop scroll	19	023	0x13	0001 0011
Ctrl - T	Device control 4	20	024	0x14	0001 0100
Ctrl - U	Negative acknowledgment	21	025	0x15	0001 0101
Ctrl - V	Synchronous idle	22	026	0x16	0001 0110
Ctrl - W	End transmission block	23	027	0x17	0001 0111
Ctrl - X	Cancel	24	030	0x18	0001 1000
Ctrl - Y	End of message, interrupt	25	031	0x19	0001 1001
Ctrl - Z	Substitute, exit	26	032	0x1a	0001 1010
Ctrl - [	Escape	27	033	0x1b	0001 1011
Ctrl - /	File separator	28	034	0x1c	0001 1100
Ctrl -]	Group separator	29	035	0x1d	0001 1101
Ctrl - ^	Record separator	30	036	0x1e	0001 1110
Ctrl - _	Unit separator	31	037	0x1f	0001 1111
	Blank, space	32	040	0x20	0010 0000
!	Exclamation point	33	041	0x21	0010 0001
"	Quotation mark, double quote	34	042	0x22	0010 0010
#	Hash mark, number sign	35	043	0x23	0010 0011
$	Dollar sign	36	044	0x24	0010 0100
%	Percent sign	37	045	0x25	0010 0101
&	Ampersand	38	046	0x26	0010 0110
'	Apostrophe, single quote	39	047	0x27	0010 0111
(	Left parenthesis	40	050	0x28	0010 1000
)	Right parenthesis	41	051	0x29	0010 1001
*	Asterisk, star, times	42	052	0x2a	0010 1010
+	Plus	43	053	0x2b	0010 1011
,	Comma	44	054	0x2c	0010 1100
-	Dash, minus	45	055	0x2d	0010 1101
.	Dot. period, decimal point	46	056	0x2e	0010 1110
/	Slash	47	057	0x2f	0010 1111
0	Digit Zero	48	060	0x30	0011 0000

تابع شفرات ASCII

Character	Description	Decimal	Octal	Hex	Binary
1	Digit one	49	061	0x31	0011 0001
2	Digit two	50	062	0x32	0011 0010
3	Digit three	51	063	0x33	0011 0011
4	Digit four	52	064	0x34	0011 0101
5	Digit five	53	065	0x35	0011 0100
6	Digit six	54	066	0x36	0011 0110
7	Digit seven	55	067	0x37	0011 0111
8	Digit eight	56	070	0x38	0011 1000
9	Digit nine	57	071	0x39	0011 1001
:	Colon	58	072	0x3a	0011 1010
;	Semicolon	59	073	0x3b	0011 1011
<	Less than	60	074	0x3c	0011 1100
=	Equal to	61	075	0x1d	0011 1100
>	Greater than	62	076	0x3e	0011 1110
?	Question mark	63	077	0x3f	0011 1111
@	Commercial at sign	64	0100	0x40	0100 0000
A	Letter capital A	65	0101	0x41	0100 0001
B	Letter capital B	66	0102	0x42	0100 0010
C	Letter capital C	67	0103	0x43	0100 0011
D	Letter capital D	68	0104	0x44	0100 0100
E	Letter capital E	69	0105	0x45	0100 0101
F	Letter capital F	70	0106	0x46	0100 0110
G	Letter capital G	71	0107	0x47	0100 0111
H	Letter capital H	72	0110	0x48	0100 1000
I	Letter capital I	73	0111	0x49	0100 1001
J	Letter capital J	74	0112	0x4a	0100 1010
K	Letter capital K	75	0113	0x4b	0100 1011
L	Letter capital L	76	0114	0x4c	0100 1100
M	Letter capital M	77	0115	0x4d	0100 1101
N	Letter capital N	78	0116	0x4e	0100 1110
O	Letter capital O	79	0117	0x4f	0100 1111

تابع شفرات ASCII

Character	Description	Decimal	Octal	Hex	Binary
P	Letter capital P	80	0120	0x50	0101 0000
Q	Letter capital Q	81	0121	0x51	0101 0001
R	Letter capital R	82	0122	0x52	0101 0010
S	Letter capital S	83	0123	0x53	0101 0011
T	Letter capital T	84	0124	0x54	0101 0100
U	Letter capital U	85	0125	0x55	0101 0101
V	Letter capital V	86	0126	0x56	0101 0110
W	Letter capital W	87	0127	0x57	0101 0111
X	Letter capital X	88	0130	0x58	0101 1000
Y	Letter capital Y	89	0131	0x59	0101 1001
Z	Letter capital Z	90	0132	0x5a	0101 1010
[	Left braket	91	0133	0x5b	0101 1011
\	Backslash	92	0134	0x5c	0101 1100
]	Right braket	93	0135	0x5d	0101 1101
^	Caret	94	0136	0x5e	0101 1110
-	Underscore	95	0137	0x5f	0101 1111
'	Accent grave	96	0140	0x60	0110 0000
a	Letter lowercase A	97	0141	0x61	0110 0001
b	Letter lowercase B	98	0142	0x62	0110 0010
c	Letter lowercase C	99	0143	0x63	0110 0011
d	Letter lowercase D	100	0144	0x64	0110 0100
e	Letter lowercase E	101	0145	0x65	0110 0101
f	Letter lowercase F	102	0146	0x66	0110 0110
g	Letter lowercase G	103	0147	0x67	0110 0111
h	Letter lowercase H	104	0150	0x68	0110 1000
i	Letter lowercase I	105	0151	0x69	0110 1001
j	Letter lowercase J	106	0152	0x6a	0110 1010
k	Letter lowercase K	107	0153	0x6b	0110 1011
l	Letter lowercase L	108	0154	0x6c	0110 1100
m	Letter lowercase M	109	0155	0x6d	0110 1101
n	Letter lowercase N	110	0156	0x6e	0110 1110

تابع شفرات ASCII

Character	Description	Decimal	Octal	Hex	Binary
o	Letter lowercase O	111	0157	0x6f	0110 1111
p	Letter lowercase P	112	0160	0x70	0111 0000
q	Letter lowercase Q	113	0161	0x71	0111 0001
r	Letter lowercase R	114	0162	0x72	0111 0010
s	Letter lowercase S	115	0163	0x73	0111 0011
t	Letter lowercase T	116	0164	0x74	0111 0110
u	Letter lowercase U	117	0165	0x75	0111 0101
v	Letter lowercase V	118	0166	0x76	0111 0110
w	Letter lowercase W	119	0167	0x77	0111 0111
x	Letter lowercase X	120	0170	0x78	0111 1000
y	Letter lowercase Y	121	0171	0x79	0111 1001
z	Letter lowercase Z	122	0172	0x7a	0111 1010
{	Left brace	123	0173	0x7b	0111 1011
|	Pipe	124	0174	0x7c	0111 1100
}	Right brace	125	0175	0x7d	0111 1101
~	Tilde	126	0176	0x7e	0111 1110
Delete	Delete, rub out	127	0177	0x7f	0111 1111

الملحق B

الكلمات المفتاحية في لغة ++C

لغة ++C لها 48 كلمة مفتاحية ، هذه الكلمات الخاصة تستخدم لتحديد التركيب اللغوي لهذه اللغة.

Keyword	Description	Example
asm	Allows information to be passed to the assembler directly يسمح بتمرير المعلومات إلى الاسمبلر مباشرة	asm ("check");
auto	Storage class for objects that exist only within their own bloch صنف تخزين للكائنات الموجودة خلال البلوك الخاص بهم	auto int n;
break	Terminates a loop or a switch statement نهاية حلقة أو الأمر swich	break;
case	Used in a switch statement to specify control expression تستخدم مع الأمر switch لتحديد أمر التحكم	switch (n/10)
catch	Specifies actions to take when an exception occurs تحدد فعل معين يجب أخذه عند حدوث استثناء معين	catch (error)
char	An integer type نوع البيانات الصحيح (أو الحرفي)	char c;
class	Specifies a class declaration يحدد اعلان الصنف	class X (...);
const	Specifies a constant definition يحدد اعلان ثابت	const int s = 32;
continue	Jumps to beginning of next iteration in a loop القفز لبداية التكرار التالي للحلقة	continue;
default	The "otherwise" case in a switch statement الحالة التلقائية في الأمر switch .	default : sum = 0;
delete	Deallocates memory allocated by a new statement اعادة تحرير ذاكرة تم تحديدها بالأمر New	delete a;
do	Specifies a do..while loop يحدد الحلقة do ... while	do { ... } while ...
double	A real number type نوع الأرقام الحقيقية	double x;
else	Specifies alternative in an if statement تحدد الخيار للأمر if	else n = 0;

تابع الكلمات المفتاحية في لغة C++

تابع الكلمات المفتاحية C++

Keyword	Description	Example
enum	Used to declare an enumeration type تستخدم للاعلان عن الأنواع المتعددة	enum tool { ... };
extern	Storage class for objects declared outside the local block صنف تخزين للكائنات المصرح عنها خارج البلوك المحلي	extern int max;
float	A real number type نوع للأرقام الحقيقية	dloat x;
for	Specifies a for loop تحدد الحلقة for	for (; ;) ...
friend	Specifies a friend function in a class تابع صديق في صنف	friend int f ( );
goto	Causes execution to jump to a labeled statement يسبب انتقال التنفيذ إلى تعليمة معنونة بلافتة	goto error;
if	Specifies an if statement تحدد الأمر if	if (n > o) ...
inline	Declares a function whose text is to be substituted for its call تحدد تابع يتم احلال نصها عند النداء عليها	inline int f ( );
int	An integer type نوع الأرقام الصحيحة	int n;
long	Used to define integer and real types يحدد أرقام صحيحة وحقيقية	long double x;
new	Allocates memory يحجز ذاكرة	int* p = new int;
operator	Used to declare an operator overload يستخدم للاعلان عن زيادة تحميل معامل	X operator ++ ( );
private	Specifies private declarations in a class يحدد اعلانات خاصة في صنف	private: int n;
protected	Specifies protected declarations in a class يحدد اعلانات محمية في صنف	protected: int n;
public	Specifies public declarations in a class يحدد اعلانات عامة في صنف	public: int n;
register	Storage class specifier for objects stored in registers محدد تخزين لتحديد كائن تخزن في مسجل	register int i;

تابع الكلمات المفتاحية C++

Keyword	Description	Example
return	Statement that terminates a function and returns a value أمر ينهي تابع ويعود بقيمة	return 0;
short	An integer type نوع أرقام صحيحة	short n;
signed	Used to define integer types يستخدم لتحديد ارقام صحيحة	signed char c;
sizeof	Operator that returns the number of bytes used to store an objects معامل يعود بعدد البايتات المستخدمة في تخزين الكائن .	n = sizeof (float;
static	Storage class of objects that exist for the duration of the program نوع تخزين يكون موجود أو معرف طالما البرنامج موجود	static int n;
struct	Specifies a structure definition اعلان تحديد بنية	struct X { ... };
switch	Specifies a switch statement يحدد الأمر swich	switch (n) { ... };
template	Specifies a template class يحدد صنف نموذج	template <class T>
this	Pointer that points to the current object مؤشر للكائن الحالي	return *this;
throw	Used to generate an exception يستخدم لتوليد استثناء	throw X ();
try	Specifies a block that contains exception handlers تحدد بلوك يحتوي على مناول استثناء	try { ... }
typedef	Declares a synonym for an existing type يعلن عن مرادف لنوع موجود	typedef int Num;
union	Specifies a structure whose elements occupy the same storage يحدد بنية تحتل كل عناصرها نفس المخزن .	\union z { ... } ;
unsigned	Used to define integer types يستخدم لتعريف أنواع أرقام صحيحة	unsigned int b;
virtual	Declares a member function that is defined in a subclass يعلن عن تابع عضو معرف في صنف فرعي	virtual int f ();
void	Designates the absence of a type يميز غياب النوع	void f ();

تابع الكلمات المفتاحية ++C

Keyword	Description	Example
volatile	Declares objects that can be modified outside of program control يعلن عن كائنات يمكن تعديلها خارج نطاق تحكم البرنامج.	int volatile n;
while	Specifies a while loop يحدد الحلقة while	while (n > 0) ...

الملحق C

العمليات في لغة C++

هذا الجدول يبين جميع العمليات في لغة C++ مرتبة على حسب أسبقية العمليات حيث العمليات ذات الأسبقية الأعلى تنفذ قبل العمليات ذات الأسبقية الأقل. فمثلاً في العلاقة (a - b*c) ، عملية الضرب * ستنفذ أولاً ثم عملية الطرح ثانياً لأن عملية الضرب لها أسبقية أعلى (13) من عملية الطرح (12). العمود المعنون "Assoc." (وهي اختصار لكلمة إلحاق Association) يخبرنا إذا كانت العملية تلحق من اليمين أم من اليسار، أما العمود المعنون "Arity" فيخبرنا عما إذا كانت هذه العملية تجري على معامل واحد أم معاملان أم ثلاثة. العمود المعنون "Ovrldbl." يخبرنا إذا كانت هذه العملية يمكن زيادة تحميلها أم لا.

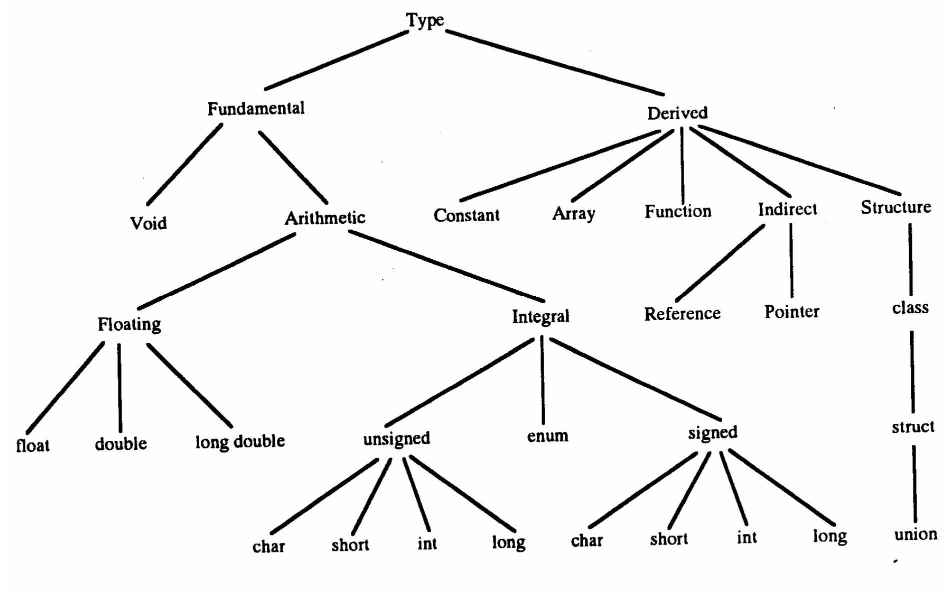
Op	Name	Prec.	Assoc.	Arity	Ovrldbl.	Example
: :	Global scope resolution	17	Right	Unary	No	: : x
: :	Class scope resolution	17	Left	Binary	No	x : : x
.	Direct member selection	16	Left	Binary	No	S. len
->	Indirect member selection	16	Left	Binary	Yes	p-> len
[ ]	Subscript	16	Left	Binary	Yes	a [i]
( )	Function call	16	Left	n\a	Yes	rand ( )
( )	Type construction	16	Left	n/a	Yes	int (ch)
++	Post-increment	16	Right	Unary	Yes	n++
- -	Post-decrement	16	Right	Unary	Yes	n--
sizeof	Size of object or type	15	Right	Unary	No	sixeof (a)
++	Pre-increment	15	Right	Unary	Yes	++n
- -	Pre-decrement	15	Right	Unary	Yes	--n
~	Bitwise complement	15	Right	Unary	Yes	~s
!	Logical NOT	15	Right	Unary	Yes	! p
+	Unary plus	15	Right	Unary	Yes	+n
-	Unary minus	15	Right	Unary	Yes	-n
*	Dereference	15	Right	Unary	Yes	*p
&	Address	15	Right	Unary	Yes	& x
new	Allocation	15	Right	Unary	Yes	new p
delete	Deallocation	15	Right	Unary	Yes	delete p
( )	Type conversion	15	Right	Binary	Yes	int (ch)
. *	Direct membeer selection	14	Left	Binary	No	x. * q

أسبقية العمليات في لغة C++

Op	Name	Prec.	Assoc.	Arity	Ovrldbl.	Example
->*	Indirect membeer selection	14	Left	Binary	Yes	p-> q
*	Multiplication	13	Left	Binary	Yes	m*n
/	Division	13	Left	Binary	Yes	m/n
%	Remainder	13	Left	Binary	Yes	m%n
+	Addition	12	Left	Binary	Yes	m + n
-	Subtraction	12	Left	Binary	Yes	m - n
<<	Bit shift left	11	Left	Binary	Yes	cout << n
>>	Bit shift right	11	Left	Binary	Yes	cin >> n
<	Less than	10	Left	Binary	Yes	x < y
<=	Less than or equal to	10	Left	Binary	Yes	x <= y
>	Greater than	10	Left	Binary	Yes	x > y
>=	Greater than or equal to	10	Left	Binary	Yes	x >= y
= =	Equal to	9	Left	Binary	Yes	x = = y
! =	Not equal to	9	Left	Binary	Yes	x ! = y
&	Bitwise AND	8	Left	Binary	Yes	s&t
^	Bitwise XOR	7	Left	Binary	Yes	s^t
|	Bitwise OR	6	Left	Binary	Yes	s | t
&&	Logical AND	5	Left	Binary	Yes	u && v
||	Logical OR	4	Left	Binary	Yes	u || v
? :	Conditional expression	3	Left	Tamary	No	u ? x : y
=	Assignment	2	Right	Binary	Yes	n = 22
+ =	Addition assignment	2	Right	Binary	Yes	n + = 8
- =	Subtraction assignment	2	Right	Binary	Yes	n - = 4
* =	Multiplication assignment	2	Right	Binary	Yes	n * = -1
/ =	Division assignment	2	Right	Binary	Yes	n \ = 10
% =	Remainder assignment	2	Right	Binary	Yes	n %= 10
& =	Bitwise AND assignment	2	Right	Binary	Yes	s &= mask
^ =	Bitwise XOR assignment	2	Right	Binary	Yes	s ^= mask
| =	Bitwise OR assignment	2	Right	Binary	Yes	s |= mask
<<=	Bit shift left assignment	2	Right	Binary	Yes	s <<= 1
>>=	Bit shift right assignment	2	Right	Binary	Yes	s >>= 1
throw	Throw exception	1	Right	Unary	Yes	throw
,	Comma	0	Left	Binary	Yes	++m, --n

الملحق D

الأنواع في لغة C++



الملحق E

توابع لغة C++ مسبقة التعريف

هذا الملحق يصف التوابع مسبقة التعريف الموجودة في مكتبة C++ . العمود الأول يحتوي اسم التابع والثاني يحتوي النموذج الأولي لهذا التابع ووصف مختصر لما يفعله ، والعمود الثالث يبين اسم الملف الرأسي المعرف فيه هذا التابع.

Function	Prototype and Description	Header File
abort ( )	void abort ( ) ; خروج من البرنامج	<stdlib. h>
abs ( )	int abs (int n) ; يعود بالقيمة المطلقة لـ n	<stdlib. h>
acos ( )	double acos (double x) ; يعود بعكس الـ cosine	<math. h>
asin ( )	double asin (double x) ; يعود بعكس الـ sin	<math. h>
atan ( )	double atan (double x) ; يعود بعكس tangent	<math. h>
atof ( )	double atof (const char* s); يعود بالرقم الحقيقي الممثل حرفياً في السلسلة S	<stdlib. h>
atoi ( )	int atoi (const char* s) ; يعود بالرقم الصحيح الممثل في السلسلة S	<stdlib. h>
atol ( )	long atol (const char* s) ; يعود بالرقم الصحيح الممثل في السلسلة S	<stdlib. h>
bad ( )	int ios : : bad ( ) ; يعود بقيمة غير الصفر إذا كانت badbit تساوي صفر ، ويعود بصفر اذا كانت غير ذلك	<iostream. h>
bsearch ( )	void* bsearch (const void* x, void* a, size_t n, size_t s, int (*cmp) (const void*, const void*)) ;	<stdlib. h>

تتمة توابع لغة C++ مسبقة التعريف

Function	Prototype and Description	Header File
	ينفذ خوارزيزم البحث الثنائي للبحث عن x في الصف المرتب a الذي يحتوي n من العناصر كل منها له حجم S مستخدماً comp* لمقارنة أي عنصرين. إذا وجد العنصر يعود بمؤشر إليه ، وإذا لم يوجد يعود المؤشر بصفر.	
ceil ( )	double ceil (double x) ; يعود بـ x مقربة لأقرب رقم صحيح	<math. h>
clear ( )	void ios : : clear (int n=0) ; يغير حالةالمجرى إلى n	<iostream. h>
clearerr ( )	void clearerr (FILE* p) ; يصفر العلم end-of-file و error للملف P*	<stdio. h>
close ( )	void fstreambase : : close ( ); **يغلق الملف الملحق للكائن القاعدي**	<fstream. h>
cos ( )	double cos (double x); Returns the inverse cosine of x.	<math. h>
cosh ( )	double cosh (double x) ; Returns the hyperbolic cosine of x: $(e^x + e^{-x})/2$.	<math. h>
difftime ( )	double difftime (time_t t1, time_t t 0) يعود بالزمن بالثواني بين t0 و t1	<time. h>
eof ( )	int ios : : eof ( ) ; يعود بقيمة غير الصفرإذا كانت eofbit=1 ويعود بالصفر في غير ذلك .	<iostream. h>
exit ( )	void exit (int n) ; ينهي البرنامج ويعود بـ n للبرنامج المنادي	<stdlib. h>
exp ( )	double exp (double x) ; returns the exponential of z: e^x .	<math. h>
fabs ( )	double fabs (double x) ; Returns the absolute value of x.	<math. h>
fail ( )	int ios : : fail ( ) ; يعود بقيمة غير الصفر إذا كانت failbit=1 ويعود بصفر إذا كانت غير ذلك	<iostream. h>
fclose ( )	int fclose (FILE* p)	<stdio. h>

تتمة توابع لغة C++ مسبقة التعريف

Function	Prototype and Description	Header File
	يغلق الملف P* ويفرغ كل العوازل. يعود بصفر إذا تمت هذه العملية بنجاح، ويعود بـ EOF في غير ذلك	
fgetc ( )	int fgetc (FILE* p) ; يقرأ ويعود بالحرف التالي من الملف P* وإلا فانه يعود بـ EOF	<stdio. h>
fgets ( )	char* fgets (char* s, int n, FILE* p); يقرأ السطر التالي من الملف P* ويخزنه في S* . الحرف الفارغ NULL يلحق بالحروف المخزنة في S . إذا لم تنجح العملية يعود بـ NULL	<stdio. h>
fill ( )	char ios : : fill ( ) يعود بحرف الحشو الحالي	<iostream. h>
	char ios : : fill (char c) ; يغير حرف الحشو الحالي إلى C ويعود بحرف الحشو السابق	
flags ( )	long ios : : flags ( ); يعود بأعلام التشكيل الحالية long ios : : flags (long n); يغير اعلام التشكيل الحالية إلى n ويعود بالاعلام السابقة	<iostream. h>
floor ( )	double floor (double x) ; يقرب x إلى أقرب أقل رقم صحيح	<math. h>
flush ( )	ostream& ostream : : flush ( ) ; يفرغ مخازن الخروج ويعود بالمجاري الجديدة	<iostream. h>
fopen ( )	FILE* fopen (const char* p, const char* s) ; يفتح الملف P* ويعود بعنوان الهيكل الممثل للملف إذا نجح، ويعود بـ NULL في غير ذلك. السلسلة S تحدد حالة الملف: "r" للقراءة و "w" للكتابة ، و "a" للإلحاق ، و "r+" للقراءة والكتابة في ملف موجود ، و "w+" للقراءة والكتابة في ملف موجود ، و "a+" للقراءة والالحاق لملف موجود	<stdio. h>
fprintf ( )	int fprintf (FILE* p, const char* s, . . .); تكتب خروج مشكلة للملف P* . تعود بعدد الاحرف التي كتبت اذا نجحت العملية، والا فإنها تعود بقيمة سالبة	<stdio. h>
fputc ( )	int fputc (int c, FILE* p); تكتب الحرف C في الملف P* . يعود بالحرف المكتوب وإلا فإنه يعود بـ EOF في حالة عدم النجاح	<stdio. h>

تتمة توابع لغة C++ مسبقة التعريف

Function	Prototype and Description	Header File
fputs ( )	int fputs (const char* s, FILE* p); يكتب السلسلة S في المف P* . يعود بقيمة غير سالبة إذا نجح وإلا فانه يعود بالقيمة EOF	<stdio. h>
fread ( )	size_t fread (void* x, size_t k, size_t n, FILE* p); يقرأ حتى عدد n عنصر له الحجم k من الملف P* ويخزنهم عند الموضع S في الذاكرة . يعود بعدد العناصر المقروءة	<stdio. h>
fscanf ( )	int fscanf (FILE* p, const char* s, ...) ; يقرأ دخل مشكل من الملف P* ويخزنهم عند الموضع S في الذاكرة. يعود بـ EOF إذا وصل نهاية الملف وإلا فإنه يعود بعدد العناصر المقروءة	<stdio. h>
fseek ( )	int fseek (FILE* p, long k, int base) ; Repositions the position marker of the file *p k bytes from its base, where base should be SEEK_SET for the beginning of the file, SEEK_CUR for the current position of the file marker, or SEEK_END for the end of the file. Returns 0 if successsful	<stdio. h>
ftell ( )	long ftell (FILE* p) ; يعود بموضع علامة مكان الملف P* وإلا فانه يعود بـ 1-	<stdio. h>
fwrite ( )	size_t fwrite (void* s, size_t k, size_t n, FILE* p) ; يكتب n من العناصر كل منها له حجم k للملف P* ويعود بالرقم الذي تمت كتابته	<stdio. h>
gcount ( )	int istream : : gcount ( ) ; يعود بأخر الحروف التي تمت قراءتها	<iostream. h>
get ( )	int istream : : get ( ) ; istream& istream : : get (signed char& c); istream& istream : : get (unsigned char& c) ; istream& istream : : get (signed char* b, int n, char e = '\ n ') ; istream& istream : : get (unsigned char* b, int n, char e= '\ n ') ;	<iostream. h>

تتمة توابع لغة ++C مسبقة التعريف

Function	Prototype and Description	Header File
	يقرأ الحرف التالي C من مجرى الدخل . الاصدار الأول يعود بـ C أو EOF . الاصداران الأخيران يقران حتى n من الحروف في b ويقف عندما يجد e	
getc ()	int getc (FILE* p) ; Same as fgetc ( ) except implemented as a macro.	<stdio. h>
getchar ( )	int getchar ( ) ; يعود بالحرف التالي من مدخل قياسي وإلا فإنه يعود بـ EOF	<stdio. h>
gets ( )	char* gets (char* s) ; يقرأ الحرف التالي من مدخل قياسي ويخزنه في S . يعود بـ S أو NULL إذا لم يقرأ أي حرف	<stdio. h>
good ( )	int ios : : good ( ) ; يعود بقيمة غير الصفر اذا كانت حالة المجرى صفر وإلا فإنه يعود بصفر	<iostream. h>
ignore ( )	istream& ignore (int n=1, int e=EOF); يستخلص حتى عدد n من الحروف من المجرى ، أو حتى الحرف e ، ايهما يأتي أولاً . يعود بالمجرى	<iostream. h>
isalnum ( )	int isalnum (int c) ; يعود بقيمة غير الصفر إذا كان e حرف هجاء أو رقم وإلا فإنه يعود بصفر	<ctype. h>
isalpha ( )	int isalpha (int c); يعود بقيمة غير الصفر إذا كان c حرف هجاء ، وإلا فإنه يعود بصفر	<ctype. h>
iscntrl ( )	int iscntrl (int c) ; يعود بقيمة غير الصفر إذا كان c حرف تحكم ، وإلا فإنه يعود بصفر	<ctype. h>
isdigit ( )	int isdigit (int c) ; يعود بقيمة غير الصفر إذا كان c رقم وإلا فإنه يعود بصفر	<ctype. h>
isgraph ( )	int isgraph (int c); يعود بقيمة غير الصفر إذا كان c حرف كتابة غير فارغ ، وإلا فانه يعود بصفر	<ctype. h>
islower ( )	int islower (int c) ; يعود بقيمة غير الصفر إذا كان c حرف صغير ، وإلا فإنه يعود بصفر	<ctype. h>
isprint ( )	int isprint (int c) ; يعود بقيمة غير الصفر إذا كان c حرف قابل للطباعة ، وإلا فإنه يعود بصفر	<ctype. h>
ispunct ( )	int ispunct (int c) ; يعود بقيمة غير الصفر إذا كانت c حرف تشكيل وإلا فإنه يعود بصفر	<ctype. h>

تتمة توابع لغة ++C مسبقة التعريف

Function	Prototype and Description	Header File
isspace ( )	int isspace (int c); يعود بقيمة غير الصفر إذا كان c أي حرف أبيض بما في ذلك الحروف ' ' و '\f' و '\n' و '\t' و '\v' وإلا فإنه يعود بصفر	<ctype. h>
isupper ( )	int isupper (int c) ; يعود بقيمة غير الصفر إذا كان c حرف كبير ، وإلا فإنه يعود بصفر	<ctype. h>
isxdigit ( )	int isxdigit (int c) ; يعود بقيمة غير الصفر إذا كان c واحد من العشرة أرقام أو واحد من الـ 12 رقم الستعشري ، وإلا فإنه يعود بصفر	<ctype. h>
labs ( )	long labs (long n) ; يعود بالقيمة المطلقة لـ n	<stdlib. h>
log ( )	double log (double x) ; Returns the natural logarithm (base e) of x.	<math. h>
log10 ( )	double log (double x) ; Returns the natural logarithm (base 10) of x.	<math. h>
memchr ( )	void* memchar (const void(s, int c, size_t k) ; يبحث العدد k من البايتات التي تبدأ عند S عن الحرف c . إذا وجد يعود بعنوان أول حدوث له ، وإلا فإنه يعود بـ NULL	<string. h>
memcmp ( )	int memcmp (const void* s1, const void* s2, size_t k); يقارن k بايت من الذاكرة تبدأ عند S1 مع k بايت من الذاكرة تبدأ عند S2 ويعود برقم سالب أو صفر أو موجب على حسب إذا كانت السلسلة الأولى أقل من أو تساوي أو أكبر من السلسلة الثانية	<string. h>
memcpy ( )	void* memcpy (const void* s1, const void* s2, size_t k) ; ينسخ الـ k بايت من الذاكرة والتي تبدأ عند S2 في مكان الذاكرة S1 ، ويعود بـ S1	<string. h>
memmove( )	int memmove (const void* s1, const void* s2, size_t k) ; مثل السابقة فيما عدا امكانية تداخل السلاسل	<string. h>
open ( )	void fstream : : open (const char* f, int m,	<fstream. h>

تتمة توابع لغة C++ مسبقة التعريف

Function	Prototype and Description	Header File
	int p=filebuf : : openprot) ; void ifstream : : open (const char* f, int m=ios : : in, int p=filebuf : : openprot) ; void ofstream : : open (const char* f, int m=ios : : out, int p=filebuf : : openprot) ; يفتح الملف f في الحالة m والحماية p	
peek ( )	int istream : : peek ( ); يعود بالحرف التالي أو الـ EOF من المجرى دون استخلاصه	<iostream. h>
pow ( )	double pow (double x, double y) ; Returns x raised to the power y (x^y) .	<math. h>
precision ( )	int ios : : precision ( ) ; int ios : : precision (int k) ; يعود بالدقة الحالية للمجرى ، الصورة الثانية تغير الدقة الحالية إلى k وتعود بالدقة القديمة	<iostream. h>
tolower ( )	int tolower (int c) ; تعود بالصورة الصغيرة للحرف c إذا كان على الصورة الكبيرة ، وإلا فإنها تعود بالحرف c	<ctype. h>
toupper ( )	int toupper (int c) ; تعود بالصورة الكبيرة للحرف c إذا كان على الصورة الصغيرة ، وإلا فإنها تعود بالحرف c	<ctype. h>

المراجع العربية

1- جون ر. هيوبارد : ترجمة: أ.د. محمد إبراهيم العدوي وآخرون، سلسلة شوم. البرمجة بلغة C++ . الجزء الأول – الدار الدولية للاستثمارات الثقافية ، القاهرة ، مصر ، 2000.

2- جون ر. هيوبارد ، ترجمة : أ.د. محمد إبراهيم العدوي وآخرون : سلسلة شوم. البرمجة بلغة C++ . الجزء الثاني. الدار الدولية للاستثمارات الثقافية ، القاهرة، مصر، 2000.

3- ترجمة مركز التعريب والبرمجة، C++ الدليل الكامل، دروس إضافية على الوب مع شهادة علمية ، الدار العربية للعلوم ، بيروت ، لبنان 1997.

4- جسي لبيرتي ، ترجمة بلال عبد القادر ، تعلم C++ في 24 درساً ، أكاديميا انترناشيونال ، بيروت ، لبنان 1998.

5- ترجمة د. صلاح الدوه جي ، كيف تبرمج بلغة C++ ، دار شعاع للنشر والعلوم ، حلب، سوريا ، 1999.

6- د. مأمون نعيم : كيف نبرمج بلغة visual C++.net ، دار شعاع للنشر والعلوم. حلب ، سوريا ، 2004.

7- د. إبراهيم نائب و د. محمد نور برهان : أساسيات البرمجة بلغة C++ ، دار وائل للنشر والتوزيع. عمان، الأردن، الطبعة الأولى ، 2003.

8- د. فادي فوز : البرمجة والحاسبات : منشورات جامعة حلب، 1988.

9- د. م. سليمان : حاسبات C++(1) ، مديرية الكتب والمطبوعات ، جامعة تشرين ، سوريا 2002 .

المراجع الأجنبية

1. Amarican national standard. Programming language C++. NY: Washington, 2000.
2. Anderson, A.E., and W.J. Heinze, C++ programming and fundamental concepts, Englewood Cliffs, NJ: Prentice-Hall, 1992.
3. H. M. Deitel and P.J. Deitel : C++ How to program, prentice Hall Internationa, Inc. (U.S.A.) 1994.
4. H. M. Deitel and P.J. Deitel : Procedural and object - Oriented programming, NJ: prentice – Hall 1995.
5. H. M. Deitel and P.J. Deitel: C++ How to program, prentice Hall International, Inc. (U.S.A.) 2005.
6. Kenneth A. Barclay and Brian J. Gordon: C++ problem solving and programming, prantice hall International (UK) Limited ,1994.
7. Plum, T., and D. Saks. C++ programming guide lines, lowrence, KS: Plum Hall 1991.
8. The ANSI/ISO, Danny Kalev: C++ professional programmer's Handbook, Que corporation (U.S.A.) 1999.
9. wirfs-brock, R., B. Wilkerson, and L. Wiener. Designing object-oriented software, Englewood Cliffs, NJ: prentice-Hall, 1990.

المصطلحات العلمية

A

الوصول	Access

جمع	Add
عنوان	Address
خوارزمية	Algorithm
تحديد	Allocation
أبجدي رقمي	Alphanumeric
عامل منطقي و	And
إضافة	Append
باراميتر (وسيط)	Argument
حسابي	Arithmetic
تعبير حسابي	Arithmetic expression
وحدة الحساب والمنطق	Arithmetic logic unit
معامل حسابي	Arithmetic operand
مصفوفة	Array
عنصر مصفوفة	Array element
لغة التجميع	Assembly language
يخصص	Assign
تعيين، تخصيص	Assignment
صفات	Attributes
متوسط	Average
آلي، تلقائي، أتوماتيكي	Auto

B

حالة أساسية (قاعدية)	Base case
سلوك، اقفال	Behaviors
ثنائي	Binary
ترميزة ثنائية	Binary code
العامل الثاني لتحديد مجال الرؤية	Binary scope resolution operator
رقم ثنائي (بت)	Binary digit (bit)
خانة – بت	Bit
فراغ	Blank
مجموعة (بلوك)	Block
منطقي	Boolean
التصنيف الفقاعي	Bubble sort
بايت (8 خانة)	Byte

C

استدعاء	Call
حالة	Case
وحدة المعالجة المركزية	Central Processing Unit (C.P.U)
رمز	Character
تغيير	Change
الأصناف	Classes
شيفرة	Code
تعليق، ملاحظة	Comment

يقارن	Compare
عوامل المقارنة	Comparison operators
مترجم	Compiler
حاسوب	Computer
حالة (شرطة مقارنة)	Condition
ثابت	Constant
تحكم	Control
نسخ	Copy
الحالي	Current

D

معطيات ، بيانات	Data
نوع البيانات	Data type
افتراضي	Default
تصريح – إعلان	Declaration
تعريف	Definition
هابط – نازل	Descendant
مشتق	Derived
تنازلي	Descending
خانة – رقم	Digit
بعد	Dimension
مضاعف الدقة – حقيقي	Double

حلقة do-while	Do-while

E

خطأ	Error
تنفيذي	Executable
تنفيذ	Execution
أسي	Exponential
تعبير	Expression
خارجي	External

F

عاملي	Factorial
حقل	Field
ملف	File
حقيقي	Float
قاعدة النقطة العائمة	Floating-point-notation
مخطط سير العمليات	Flow-chart
تعليمة التكرار for	For
واصف	Format
لغة فورتران	Fortran
تابع	Function

G

عام	General

عالمي – شائع	Global
متغير عام – شائع	Global variable
تعليمة الانتقال غير المشروط	Goto

H

الكيان الصلب	Hardware
رأسي	Header
لغة عالية المستوى	High level language

I

معرف	Identifier
تطبيق، تحقيق	Implementation
تضمين	Include
زيادة	Increment
دليل	Index
غير متناهي	Infinite
معلومات	Information
دخل	Input
إدخال	Insertion
اشتقاق أو خلق كائن	Instantiation
واجهة	Interface
صحيح	Integer (int)
داخلي	Internal

Interpreter	المفسر
Invoking	ستحضار
Iteration	تكرار

K

Key-word	كلمة مفتاحية

L

Label	لافتة
Language	لغة
Level	مستوى
Link	ربط
List	لائحة
Local	محلي
Location	موضع
Logic	منطقي
Logical expression	تعبير منطقي
Long integer	أعداد صحيحة طويلة

M

Machine language	لغة الآلة
Main	اسم التابع الرئيسي في ++C
Member	عضو

Member access specifiers	محددات الوصول إلى الأعضاء
Memory	الذاكرة
Method	طرق
Module	نموذج، وحدة

N

Name	اسم
Nested loops	حلقات متداخلة
Not	عامل منطقي (نفي)
Numerical	عددي

O

Object	غرض، كائن
Object-Oriented-Programming	البرمجة غرضية التوجه
One dimentional array	مصفوفة أحادية
Operand	مؤثر، عامل
Operation	عملية
Operator	معامل
Or	عامل منطقي (أو)
Output	خرج
Overflow	طفحان
Overload	زيادة التحميل

P

بارامیتر (وسيط)	Parameter
عام	Public
صيغة التدوين الملحق	Post fix
زيادة لاحقة	Post increment
خاص	Private
إجراء	Procedure
برنامج	Program
لغة برمجة	Programming language
النموذج الأولي	Prototype

R

سجل	Record
تعاودي	Recursion
إجراء تعاودي	Recursive procedure
مسجل	Register
علاقاتي	Relational
رجوع، عودة	Return

S

يبحث	Search
تتابع – تتالي	Sequence
تسلسلي ، تتابعي	Sequential
عدد صحيح قصير	Short int

Sort	تصنيف
Software	برمجيات
Source	مصدر
Specifier	محدد
Statement	تعليمة
Static	ساكن
Storage	تخزين
String	سلسلة محرفية
Symbol	رمز
Syntax	تركيب
Syntax error	خطأ (قواعدي)
Sub-program	برنامج فرعي

T

Tab	جدول
Table	جدول
Text	نص
Two dimensional array	مصفوفة ثنائية البعد
Type	نوع

V

Value	قيمة
Variable	متحول، متغير

W

كلمة	Word

تم تدقيق الكتاب علمياً من قبل :

الدكتور	**الدكتور**	**الدكتور**
يحيى نجار	**أحمد قصي كيالي**	**الياس طوشان**

المدقق اللغوي

الدكتور أحمد ارحيم هبو

حقوق الطبع والترجمة والنشر محفوظة
لمديرية الكتب والمطبوعات الجامعية

المطبعة الرقمية